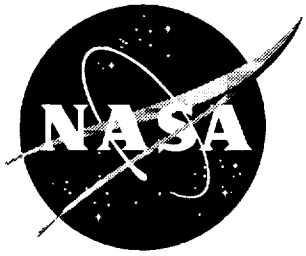
NASA/CR-97-206284



COMET-AR User's Manual

COmputational MEchanics Testbed With Adaptive Refinement

E. Moas, Editor
Applied Research Associates, Inc., Raleigh, North Carolina

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Purchase Order L-44830D

December 1997

Available from the following:

NASA Center for AeroSpace Information (CASI)
800 Elkridge Landing Road
Linthicum Heights, MD 21090-2934
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 487-4650

Preface

The COMET-AR User's Manual provides a reference manual for the COmputational MEchanics Testbed with Adaptive Refinement (COMET-AR), a software system developed jointly by Lockheed Palo Alto Research Laboratory and NASA Langley Research Center under contract NAS1-18444. The COMET-AR system is an extended version of an earlier software system called COMET (also developed by Lockheed and NASA). The primary extensions are the adaptive mesh refinement capabilities and a new "object-like" database interface that makes COMET-AR easier to extend further.

This User's Manual provides a detailed description of the user interface to COMET-AR from the viewpoint of a structural analyst. For a more concise treatment of the user interface which includes walk-through examples, see the COMET-AR Tutorial. For additional details on Adaptive Refinement (AR) theory and applications, please see the NASA Contractor Report entitled *Adaptive Refinement for Shell Structures*. For information on how to extend COMET-AR in the direction of adding new elements, new constitutive models or new data objects, consult the developer-oriented sections of the Generic Element Processor Manual, the Generic Constitutive Processor Manual, and the High-level DataBase (HDB) Manual. (See section on "Related COMET-AR Documentation" in Chapter 1 for a list of such references.)

The contents of this document were originally compiled in October 1993 by Gary Stanley of Lockheed Palo Alto Research Laboratory. Contributors include:

Contributor	Affiliation	Phone Number
Bryan HURLBUT	Lockheed Palo Alto Research Laboratory	
Itzhak LEVIT	Lockheed Palo Alto Research Laboratory	
William LODEN	Lockheed Palo Alto Research Laboratory	
Gary STANLEY	Lockheed Palo Alto Research Laboratory	(415) 424-3218
Bo STEHLIN	Lockheed Palo Alto Research Laboratory	
Lyle SWENSON	Knowledge Management Systems	

The document was edited and updated in February 1995 by Applied Research Associates, Inc. to reflect changes and additions to the system.

Version
October 1993
February 1995

Table of Contents (Brief)

Part I: INTRODUCTION

Chapter 1 Introduction

Part II: PROCEDURES

Chapter 2 Model Definition Procedures

Chapter 3 Basic Solution Procedures

Chapter 4 Adaptive Solution Procedures

Chapter 5 Utility Procedures

Part III: PROCESSORS

Chapter 6 Pre-Processors

Chapter 7 Element Processors

Chapter 8 Constitutive Processors

Chapter 9 Smoothing Processors

Chapter 10 Error Estimation Processors

Chapter 11 Mesh Refinement Processors

Chapter 12 Matrix/Vector Processors

Chapter 13 Special-Purpose Processors

Chapter 14 Post-Processors

Part IV: DATABASE

Chapter 15 Database Summary

Part V: SOLID MODEL INTERFACE

Chapter 16 Solid Model Interface (SMI)

Table of Contents (Detailed)

Part I: INTRODUCTION

Chapter 1 Introduction

1.1 Overview of COMET-AR
1.2 Purpose of This User's Manual
1.3 Capabilities and Limitations of COMET-AR
1.4 Organization of COMET-AR
1.5 Execution of COMET-AR (The User Interface)
1.6 How to Use This User's Manual
1.7 Related COMET-AR Documentation
1.8 Command Language Summary
1.9 Glossary of COMET-AR Terms, Notations, and Symbols
1.10 References

Part II: PROCEDURES

Chapter 2 Model Definition Procedures

2.1 Overview
2.2 Reference Frames and Coordinate Systems
2.3 Generic Model Definition Procedures
2.4 Node Definition Procedures
2.5 Element Definition Procedures
2.6 Material/Fabrication Definition Procedures
2.7 Orientation of Fabrication Reference Frames
2.8 Load Definition Procedures
2.9 Boundary Condition Definition Procedures
2.10 Automatic DOF Suppression and Drilling Stabilization
2.11 Sample Model Definition Procedures (Summary)
2.12 Model Definition via PATRAN and PST
2.13 Global Model to Analysis Model Translation Procedure

Chapter 3 Basic Solution Procedures

3.1 Overview
3.2 Procedure L_STATIC_1
3.3 Procedure NL_STATIC_1

Chapter 4 Adaptive Solution Procedures

4.1 Overview
4.2 Procedure AR_CONTROL

Chapter 5 Utility Procedures

5.1 Overview

5.2 Procedure ES
5.3 Procedure EST_ERR_1
5.4 Procedure EST_ERR_SM
5.5 Procedure FACTOR
5.6 Procedure FORCE
5.7 Procedure INITIALIZE
5.8 Procedure REF_MESH_1
5.9 Procedure SOLVE
5.10 Procedure STIFFNESS
5.11 Procedure STRESS
5.12 Procedure MASS

Part III: PROCESSORS

Chapter 6 Pre-Processors

6.1 Overview
6.2 Processor AUS (Nodal Force Tabulation)
6.3 Processor COP (Constraint Processor)
6.4 Processor GCP (Generic Constitutive Processor)
6.5 Processor GEP (Generic Element Processor)
6.6 Processor PST (COMET-AR_to_PATRAN)
6.7 Processor REDO (Dataset Reformatter)
6.8 Processor RENO (Node Renumbering)
6.9 Processor RSEQ (Node Renumbering)
6.10 Processor TAB (Tabulation of Nodal Coordinates)
6.11 Processor NODAL
6.12 Processor GM2AM (Geometric to Analysis Model)

Chapter 7 Element Processors

7.1 Overview
7.2 Processor ES (Generic Element Processor)
7.3 Processor ES1 (SRI and ANS Shell Elements)
7.4 Processor ES5 (STAGS Shell Element)
7.5 Processor ES6 (STAGS Beam Element)
7.6 Processor ES1p (Variable-p Lagrange Quadrilateral Shell Elements)
7.7 Processor ES7p (ANS Shell Elts.; Var. Order Quads)
7.8 Processor ES36 (MIN3/6 Triangular Shell Elements)

Chapter 8 Constitutive Processors

8.1 Overview
8.2 Generic Constitutive Processor Description
8.3 Fabrication Definition
8.4 Material Property Definition
8.5 Analysis Control
8.6 Update Command

Chapter 9 Smoothing Processors

9.1 Overview
9.2 Processor SMT (Tessler Smoothing)
9.3 Processor SMZ (Zienkiewicz/Zhu Smoothing)

Chapter 10 Error Estimation Processors

10.1 Overview
10.2 Processor ERR (Generic Error Estimator)
10.3 Processor ERR2 (Error Estimates: Stress Smoothing)
10.4 Processor ERR4 (Error Estimates: Energy Smoothing)
10.5 Processor ERR6 (Error Estimates: Strain Smoothing)
10.6 Processor ERRa (Error Accumulator)
10.7 Processor ERRSM (Error Estimates: Smoothing-Based)

Chapter 11 Mesh Refinement Processors

11.1 Overview
11.2 Processor REF1 (Mesh Refinement: $h_c/h_s/h_t/p$)

Chapter 12 Matrix/Vector Processors

12.1 Overview
12.2 Processor ASM
12.3 Processor ASMs (Matrix Assembler)
12.4 Processor ITER (Iterative Linear Equation Solver)
12.5 Processor PVSOLV (Direct Linear Equation Solver)
12.6 Processor SKY (Direct Linear Equation Solver)
12.7 Processor SKYs (Direct Linear Equation Solver)
12.8 Processor VEC (Vector Algebra Utility)
12.9 Processor VSS (Vectorized Sparse Solver)

Chapter 13 Special-Purpose Processors

13.1 Overview
13.2 AMPC (Automatic Multipoint Constraint)
13.3 Processor COMET-AR (System Macroprocessor)
13.4 Processor TRIAD (Computational Frame Realignment)

Chapter 14 Post-Processors

14.1 Overview
14.2 Processor ARG (Adaptive Refinement Graphics)
14.3 Processor HDBprt (Database Print Utility)
14.4 Processor PST

Part IV: DATABASE

Chapter 15 Database Summary

15.1 Overview
15.2 Data Objects
15.3 Database Access
15.4 Database Organization and Evolution

Part V: SOLID MODEL INTERFACE

Chapter 16 Solid Model Interface (SMI)

16.1 Overview
16.2 Solid Model Interface (SMI) Options
16.3 The Discrete SMI Option
16.4 The User-Written SMI Option

Part I

INTRODUCTION

Chapter 1 Introduction

1.1 Overview of COMET-AR

COMET-AR is an acronym for Computational Mechanics Testbed with Adaptive Refinement, a software system developed jointly by Lockheed Palo Alto Research Laboratory and NASA Langley Research Center to perform automated structural analysis of aerospace vehicles via adaptively controlled numerical simulation (i.e., finite element modeling with adaptive mesh refinement).

COMET-AR is intended to be a full-capability production code that can be utilized by a wide spectrum of structural engineers to facilitate the design of a wide variety of aerospace (and other) vehicles. Currently, it is a research code with some advanced adaptive refinement (AR) capabilities, but also with some significant gaps in generality and quality assurance. It is nonetheless capable of analyzing some very complicated problems, and has been applied to aircraft shell structural models possessing hundreds of thousands of degrees of freedom (DOF), achieving solutions that would have required many more DOFs with conventional finite element codes. We wish to make COMET-AR available to engineers who wish to benefit from its capabilities while participating in its development and evolution. This User's Manual (and the accompanying Tutorial [1]) is a prerequisite for such engineers.

The organization of this introductory chapter to the User's Manual is summarized in Table 1-1.

Table 1-1 Outline of Chapter Chapter 1: Introduction

Section	Title
1.1	Overview of COMET-AR
1.2	Purpose of This User's Manual
1.3	Capabilities and Limitations of COMET-AR
1.4	Organization of COMET-AR
1.5	Execution of COMET-AR (The User Interface)
1.6	How to Use This User's Manual
1.7	Related COMET-AR Documentation
1.8	Command Language Summary
1.9	Glossary of COMET-AR Terms, Notation, & Symbols
1.10	References

It is absolutely essential for the novice user to read Sections 1.4 and 1.5 (on organization and execution of COMET-AR, respectively) before attempting to read subsequent chapters in this manual, as these sections explain how the parts fit together into a working system.

1.2 Purpose of This User's Manual

The COMET-AR User's Manual is intended to be a reference manual for both novice and experienced COMET-AR users. The term "user" refers here to a person who wishes to employ COMET-AR to perform structural analysis without changing or adding to the existing capabilities. A typical COMET-AR user would have at least some structural analysis experience (hopefully finite element structural analysis) but little or no software development experience. (This is in contrast to a COMET-AR developer, who might be interested in developing or co-developing new capabilities within COMET-AR and who would be expected to have a software development background. The User's Manual is not intended for such a person.)

The term "reference manual" refers here to a comprehensive backup document that is used to look up information after the user is familiar with the system and acquired some hands-on experience, either with the help of the COMET-AR User's Tutorial or by a personal tutorial from an experienced colleague.

The COMET-AR User's Manual does, however, provide an overview of the system, what it can do, and how to use it, in this introductory chapter. This is no substitute for the COMET-AR Tutorial and hands-on experience. It is recommended that the prospective user read all of the current chapter, then gain experience with examples in the Tutorial. Return to the User's Manual when it is time to solve a real problem and you need to know all of the options and prepare the detailed ingredients.

1.3 Capabilities and Limitations of COMET-AR

A capabilities summary is provided in Table 1.3-1.

Table 1.3-1 Summary of COMET-AR Capabilities

Category	Capability	Description
APPLICATIONS	General Shell Structures	See, e.g., HSCT model on cover
ANALYSIS TYPES	Linear Statics	Direct and iterative equation solvers
	Nonlinear Statics	Arclength-controlled solution algorithm
ELEMENT TYPES	Quadrilateral Shell Elements	High-performance ANS formulation
	Triangular Shell Elements	High-perform. MIN3 formulation
	Beam and Solid Elements	Implemented but untested
MATERIAL MODELS	Elastic/Plastic Isotropic	White-Besseling plasticity model
	Elastic Orthotropic	2D and 3D orthotropy
	Composite Laminates	Multiple orthotropic shell layers
EXTERNAL LOADS	Point, Line, Surface, & Body	Includes live pressure loads
BOUNDARY CONDITIONS	Single-Point and Multi-Point Linear Constraints	Constraints enforced by direct elimination of superfluous unknowns
ADAPTIVE MESH REFINEMENT (AR)		
Error Estimates	Smoothing-based	Modified Zienkiewicz method
Refinement Schemes	Transition-based h (ht)	Quad. & triang. transition patterns
	Constraint-based h (hc)	Arbitrary element fission/fusion
	Uniform p	Up to p = 5 for some elements
HARDWARE	Most Serial Computers with Unix O.S.	SUN, DEC, CONVEX, CRAY, and TITAN, for example
SOFTWARE	Command-Language-Driven Fortran Processors	Solution procedures are written in high-level command language
	Modular/Extendible System Connected by Database	Developer interfaces for new elts, constit. models, and data objects

A corresponding summary of important limitations associated with each of the above categories is given in Table 1.3-2.

Table 1.3-2 Summary of COMET-AR Limitations

Category	Limitations
APPLICATIONS	Very little experience with realistic applications
ANALYSIS TYPES	Cannot perform dynamic response or eigenvalue analysis
ELEMENT TYPES	No beam or solid elements (implemented but not tested)
MATERIAL MODELS	Cannot handle finite strains
EXTERNAL LOADS	Cannot handle multiple load systems in nonlinear analysis
BOUNDARY CONDS.	Cannot handle nonlinear constraint
ADAPTIVE MESH REFINEMENT (AR)	
Error Estimates	Need to be made more robust for built-up structures
Refinement Schemes	Not compatible with all finite element types
HARDWARE	Not yet implemented on parallel processing computers
SOFTWARE	User interface is not uniformly graphical

For information on the capabilities and limitations of COMET-AR in each of these areas, refer to the appropriate chapter(s) in this User's Manual. For example, analysis types are described under the chapter on *Basic Solution Procedures*, element types, material models, external loads, and boundary conditions are described in the chapter on *Model Definition Procedures* (and other chapters referenced therein), and adaptive mesh refinement techniques are described in the chapter on *Adaptive Solution Procedures* (as well as in the chapters on *Error Estimation Processors* and *Mesh Refinement Processors*).

1.4 Organization of COMET-AR

An overview of the COMET-AR software system is shown in Figure 1.4-1. The system is modular and composed of several layers, although the user only interacts directly with the top one or two layers.

The top layer consists of command-language procedures written in a simple, high-level language called CLAMP (Command-Language for Applied Mechanics Processors). These procedures are used to control the next layer, which consists of independently executable Fortran processors. Linked into each processor are the architectural utilities: the command-language interpreter (CLIP) and the high-level database manager (HDB). At the foundation level is the database, which consists of typically one (but occasionally more) HDB files, each containing a number of datasets (which we also refer to as data objects).

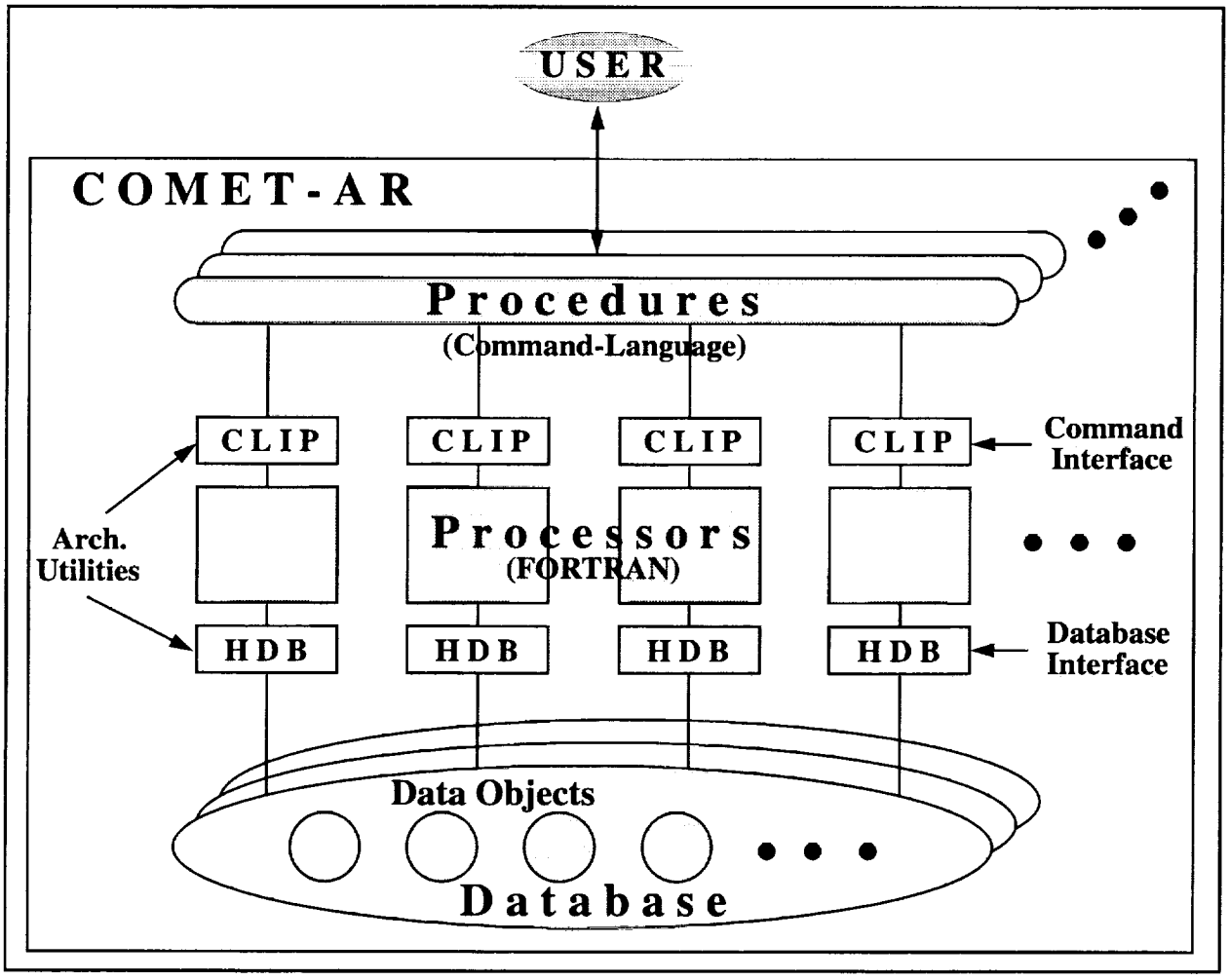


Figure 1.4-1 Overview of COMET-AR Organization

The function and capabilities available within each layer are described in the following sections.

1.4.1 COMET-AR Procedures

COMET-AR command-language procedures are either user-written or pre-defined. User-written procedures are typically employed for model definition unless an alternate pre-processor such as PATRAN is used to generate the model. Pre-defined procedures are typically employed to perform the solution. For example, basic solution procedures exist for linear and nonlinear static analysis, and a special adaptive solution procedure exists for performing linear analysis with adaptive mesh refinement. A number of other pre-defined procedures, called utility procedures, are employed by the solution procedures to perform common functions such as stiffness matrix formation, factorization, and equation solving. The utility procedures may be used to facilitate development of new solution procedures as well. A summary of currently available COMET-AR procedures is given in Table 1.4-1.

Table 1.4-1 Summary of Current COMET-AR Procedures

Procedure	Description
MODEL DEFINITION Procedures	
User-Written	Examples of these may be found in the Tutorial
BASIC SOLUTION Procedures	
L_STATIC_1	Linear static structural analysis
NL_STATIC_1	Nonlinear static structural analysis with arclength control
ADAPTIVE SOLUTION Procedures	
AR_CONTROL	Linear and nonlinear static analysis with adaptive mesh refinement
UTILITY Procedures	
ES	Performs generic element-level functions (via element processors)
EST_ERR_1	Performs error estimation functions (via error estimation processors)
FACTOR	Factors assembled matrices, e.g., stiffness (via matrix processors)
FORCE	Forms/assembles force vectors (via element and vector processors)
INITIALIZE	Initializes database prior to solution (via elt. and constraint processors)
REF_MESH_1	Performs adaptive mesh refinement (via mesh refinement processors)
SOLVE	Solves linear equation systems (via matrix processors)
STIFFNESS	Forms and assembles stiffness matrix (via elt. and assembly processors)
STRESS	Computes element stresses, strains, etc. (via element processors)

The relationship between adaptive solution procedures, basic solution procedures, and utility procedures is illustrated in Figure 1.4-2.

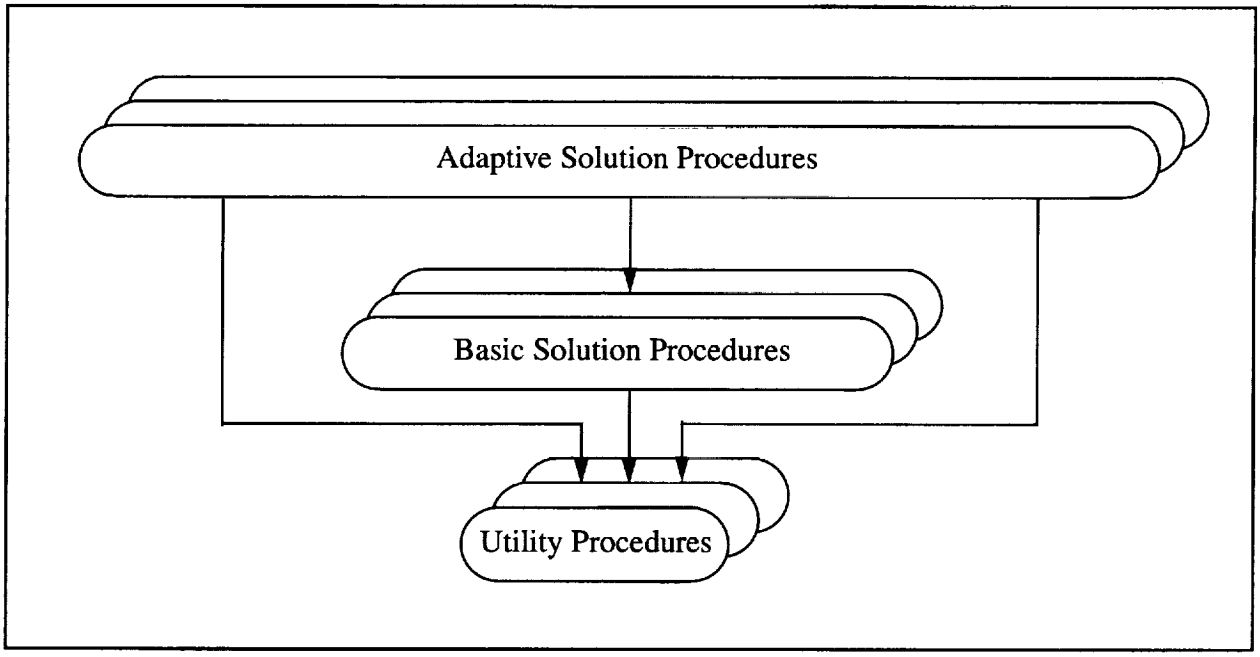


Figure 1.4-2 Hierarchy of COMET-AR Procedure Types

The same utility procedures may be used by a number of different solution procedures and the same basic solution procedures may be used by a number of different adaptive solution procedures (even though there is only one adaptive solution procedure at the moment).

Descriptions of all of the COMET-AR procedures listed in Table 1.4-1 may be found in Part II of this manual.

1.4.2 COMET-AR Processors

COMET-AR processors perform the bulk of the computational work within COMET-AR. Each processor is an independently executable module which is typically driven (i.e., orchestrated) by one or more of the COMET_AR procedures described in the preceding subsection; however, some COMET-AR processors are intended to be run interactively by the user, without intervening procedures. While it is possible for users to write new processors, it is typically not necessary unless some fundamental new capability is missing that the user can supply. A summary of currently available COMET-AR processors is given in Table 1.4-2.

Table 1.4-2 Summary of Current COMET-AR Processors

Processor	Function
Pre-Processors	
AUS	Tabulates specified nodal forces and displacements
COP	Constraint processor; tabulates boundary conditions, numbers eqns.
PST	PATRAN-to-COMET-AR conversion
REDO	Reformats certain datasets from COMET to COMET-AR format
RENO	Renumbers nodes for bandwidth optimization; geometric algorithm
RSEQ	Renumbers nodes for bandwidth optimization; variety of algorithms
TAB	Tabulates nodal coordinates and reference frame transformations
ELEMENT Processors	
ES1p	Variable-order basic Lagrange quadrilateral shell elements
ES7p	Variable-order Assumed Natural Strain (ANS) quad. shell elements
ES36	Anisoparametric MIN3/6 triangular shell elements
CONSTITUTIVE Processors	
GCP	Generic constitutive processor
ERROR ESTIMATION Processors	
ERR2	Stress-smoothing-based error estimates; Zienkiewicz's method
ERR4	Strain-energy-smoothing-based error estimates; Levit's method
MESH REFINEMENT Processors	
REF1	Adaptive mesh refinement with variety of h techniques (and uniform p)
MATRIX/VECTOR Processors	
ASM	Assembles element matrices into SKYLINE or COMPACT format
ASMs	Assembles element matrices into SKYLINE format for h_s-refinement
PVSOLV	Direct linear equation solver optimized for vector machines
SKY	Direct linear equation solver based on SKYLINE matrices
SKYs	Direct and iterative linear equation solvers for hs-refinement
ITER	Iterative linear equation solver based on COMPACT matrices
VEC	General-purpose vector algebra utility
Post-Processors	
ARGx	Interactive graphics model and solution post-processor
HDBprt	High-level database print utility
PST	COMET-AR-to-PATRAN conversion
Special-Purpose Processors	
COMET-AR	Start-up/control processor for COMET-AR software system
TRIAD	Re-aligns computational triads for automatic drilling DOF suppression

All the current COMET-AR processors are written in FORTRAN (except for parts of the common architectural utilities embedded within them), but there is no reason why new processors cannot be written in another language (such as C).

Descriptions of all of the COMET-AR processors listed in Table 1.4-2 may be found in Part III of this manual.

1.4.3 COMET-AR Architectural Utilities

Each COMET-AR processor is linked to two architectural utilities when it is created: CLIP and HDB (as illustrated in Figure 1.4-3). CLIP [2] is a command-language interpretation utility that both parses commands targeted for individual processors and executes procedure directives, special commands that appear in procedures and may be used to coordinate one or more processors (see User Interface in Section 1.5). HDB [3] is a high-level database management utility which processes most of the data objects associated with COMET-AR. HDB actually represents a conglomeration of layered database utilities. It invokes a generic database utility called DB to perform database transactions with dynamic memory management; DB in-turn invokes a name-oriented record management system called GAL [4] for all file-based data transactions.

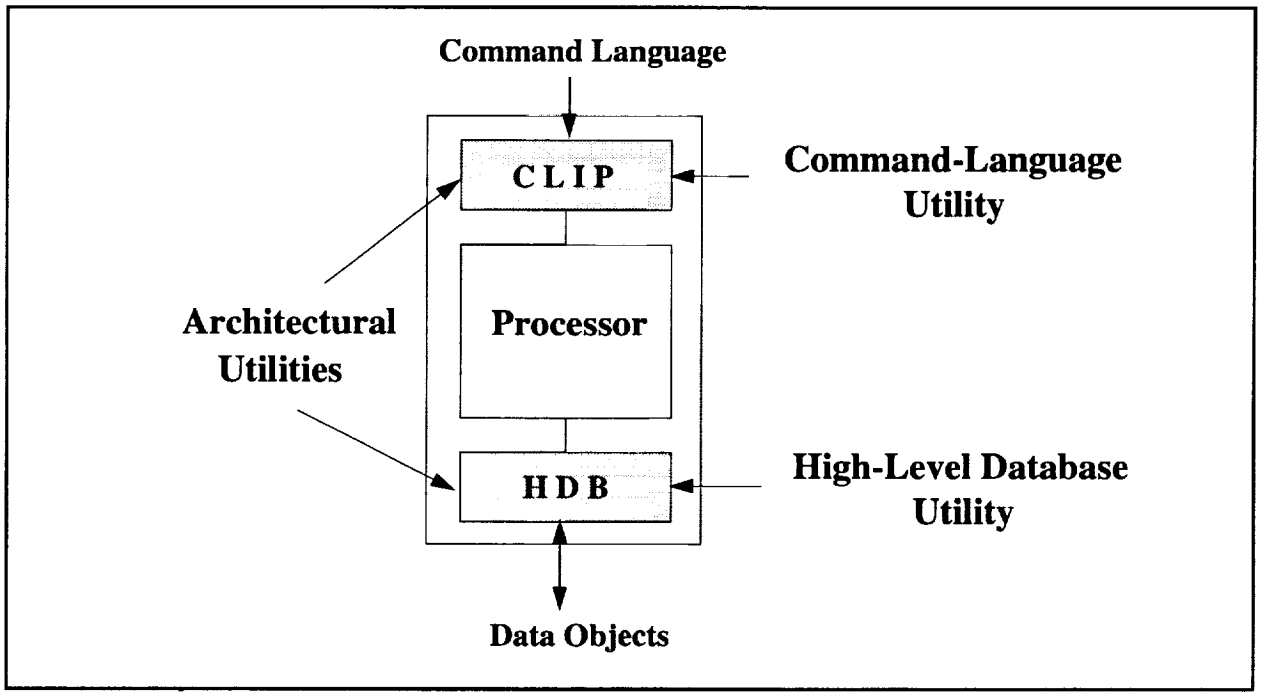


Figure 1.4-3 Relationship Between Processors and Architectural Utilities

Documentation on the special input arguments associated with each procedure, and the special commands associated with each processor, is provided in Parts II and III of this manual. Documentation on CLIP (and procedure directives) may be found in reference [2].

Documentation on the data objects associated with COMET-AR analysis is provided in Part IV of this manual. Documentation on the HDB, DB, and GAL utilities may be found in references [3], [5], and [4].

1.4.4 COMET-AR Database

The COMET-AR database (illustrated in Figure 1.4-4) consists primarily of a main (or central) disk file, typically (but not necessarily) called *Case*.DBC, where *Case* represents a user-defined case name. Such files are also called data libraries, and each contains a number of named datasets (also called data objects). Most data objects may be viewed as a table of named attributes that range over some index, such as the number of nodes in the model, the number of elements in the model, etc. (and most come equipped with their own set of Fortran access utilities to facilitate data manipulation by other code developers). Some of the data objects currently in COMET-AR contain such things as element definition parameters, element loads, element matrices, nodal coordinates, nodal vectors, system vectors, system matrices, and so on.

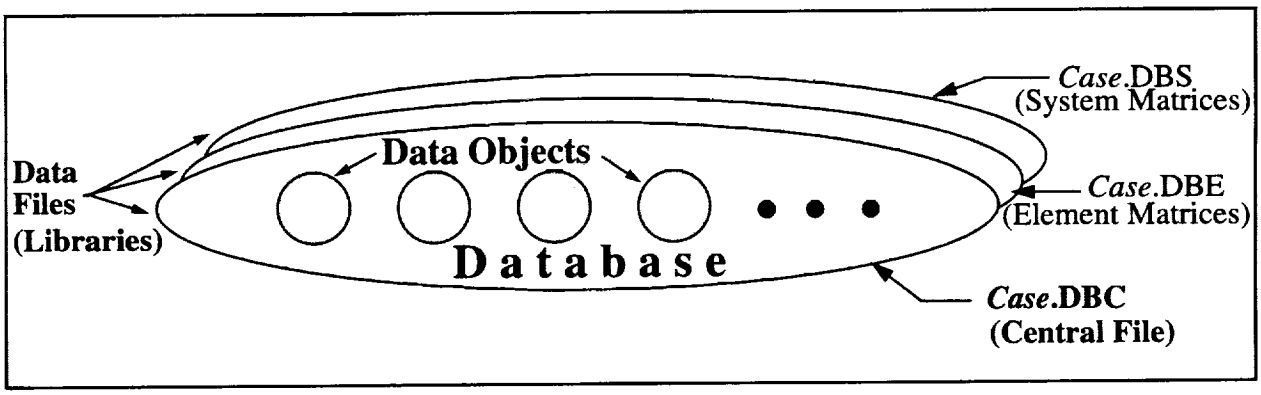


Figure 1.4-4 COMET-AR Database

In addition to the central database file (*Case*.DBC), two auxiliary files called *Case*.DBE and *Case*.DBS are often used (at the user's option in various Solution Procedures) for element and system matrices. These typically are the most space-consuming data objects, and separating them from the main database makes it easier to discard them without sacrificing any of the more user-oriented data, like displacement and stress results, which typically reside in the *Case*.DBC file.

The hierarchical structure of a COMET-AR data library may be viewed as follows.

Level 1	Level 2	Level 3	Interpretation
Data Library			File
	Dataset (= Data Object)		Records associated with node, element, or system attributes
		Record (and Record Groups)	Data associated with node, element, or system attributes

A complete description of all data objects and libraries (i.e., files) relevant to performing an analysis with COMET-AR is given in Part IV of this manual.

> COMET-AR data library (i.e., file) names such as *Case*.DBC, which appear throughout this manual, are recommended conventions. They are not mandatory. Most COMET-AR command language procedures and processors refer to the library identification number (or *ldi*) of a file, rather than to the file name, so that in general the user may choose COMET-AR file names arbitrarily.

1.5 Execution of COMET-AR (The User Interface)

1.5.1 Getting Started

Before using COMET-AR to perform a structural analysis, the following initialization steps are necessary. These steps assume the system has been installed on a computer with a UNIX operating system.

Step 1:

Modify your *.cshrc* file so that it contains the necessary PATH directions to the COMET-AR software system, as well as the necessary definition of UNIX environment variables such as *$CSM*. The proper modifications should be obtained from a representative of the COMET-AR software development staff in the Computational Mechanics Branch of NASA Langley Research Center. If COMET-AR is installed properly, this step can be accomplished by entering the UNIX-level command:

ar_login

This step only has to be performed once, preferably by the system software administrator at your installation.

Step 2:

Create a separate working directory for each new COMET-AR analysis. Copy to that directory the COMET-AR procedure library database file, called "proclib.gal." This step can be accomplished by issuing the UNIX-level command:

ar_proc

which will automatically perform the copy from the appropriate directory. If you are only using "canned" COMET-AR solution procedures and not adding any of your own, this step can be replaced by a simple soft link of the name "proclib.gal" to the actual master version of the file "proclib.gal," which should be write-protected.

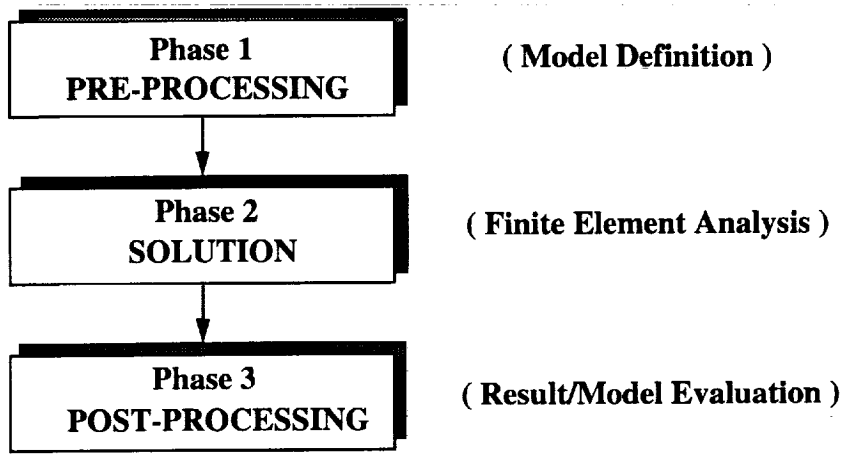
Step 3:

> Create the necessary UNIX script file(s), model definition procedures, and/or PATRAN models for the problem at hand, as described in the following subsections

The following subsections describe the process of performing an analysis with COMET-AR, from pre-processing through post-processing.

1.5.2 User Interface Overview

A COMET-AR analysis (or simulation, depending on your perspective) consists of three phases.



Each of these phases involves a somewhat different user interface, especially if PATRAN is used for pre/post-processing, as illustrated in Figure 1.5-1.

In Figure 1.5-1, the *.com* files are UNIX script files containing COMET-AR procedure calls, the *.clp* files are COMET-AR command-language procedures, and GUI denotes a graphical user interface. The ingredients for each phase are explained in detail in the following subsections.

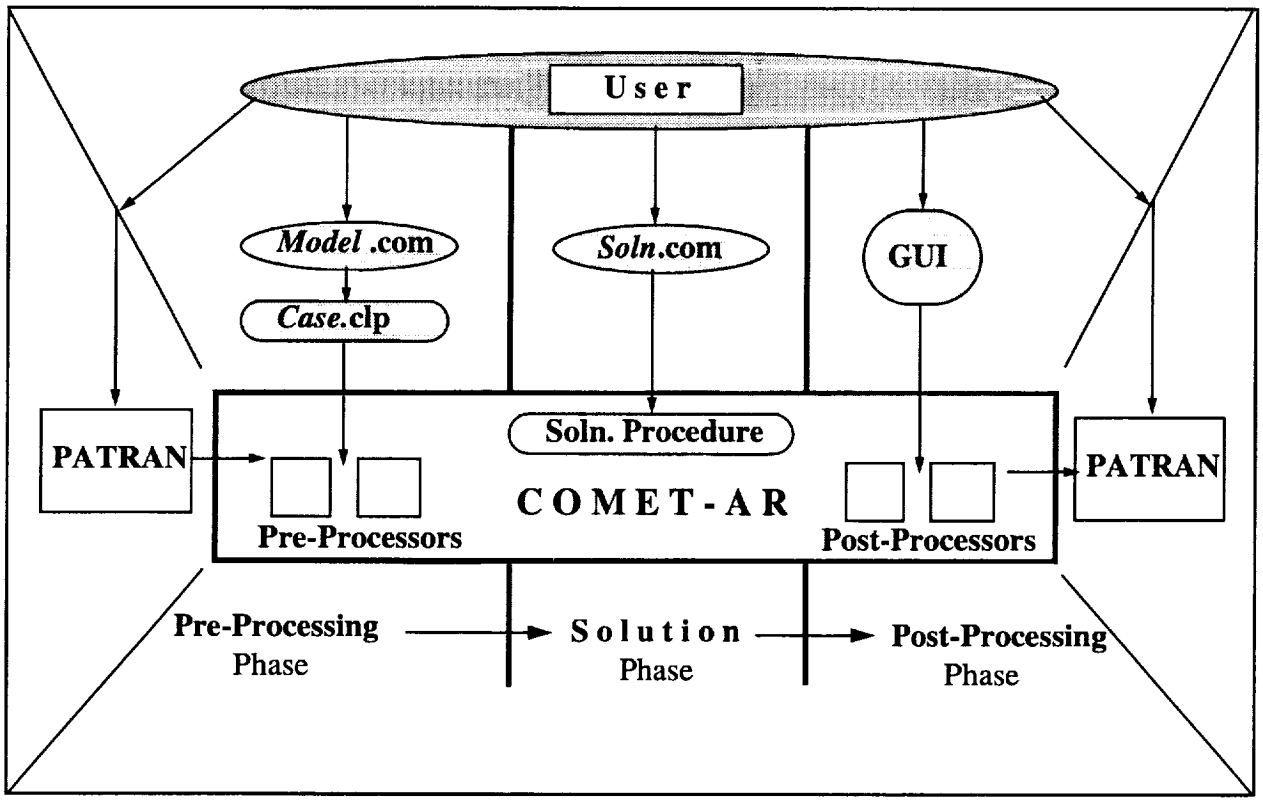


Figure 1.5-1 Overview of User Interfaces Involved in Different Analysis Phases

1.5.3 Pre-Processing Phase — Model Definition

In the Pre-Processing Phase, an initial finite element model (i.e., nodes, element types, connectivity, loads, boundary conditions, material properties, etc.) is defined by the user and stored in a COMET-AR database. The user interface for this phase depends on whether or not PATRAN is being employed to generate the initial finite element model; thus, we shall consider the two cases separately.

1.5.3.1 Pre-Processing Without PATRAN

This is currently the recommended way to define a COMET-AR initial model (as the PATRAN interface is a recent addition that is not yet considered robust). The user interface requirements for model definition are shown in Figure 1.5-2.

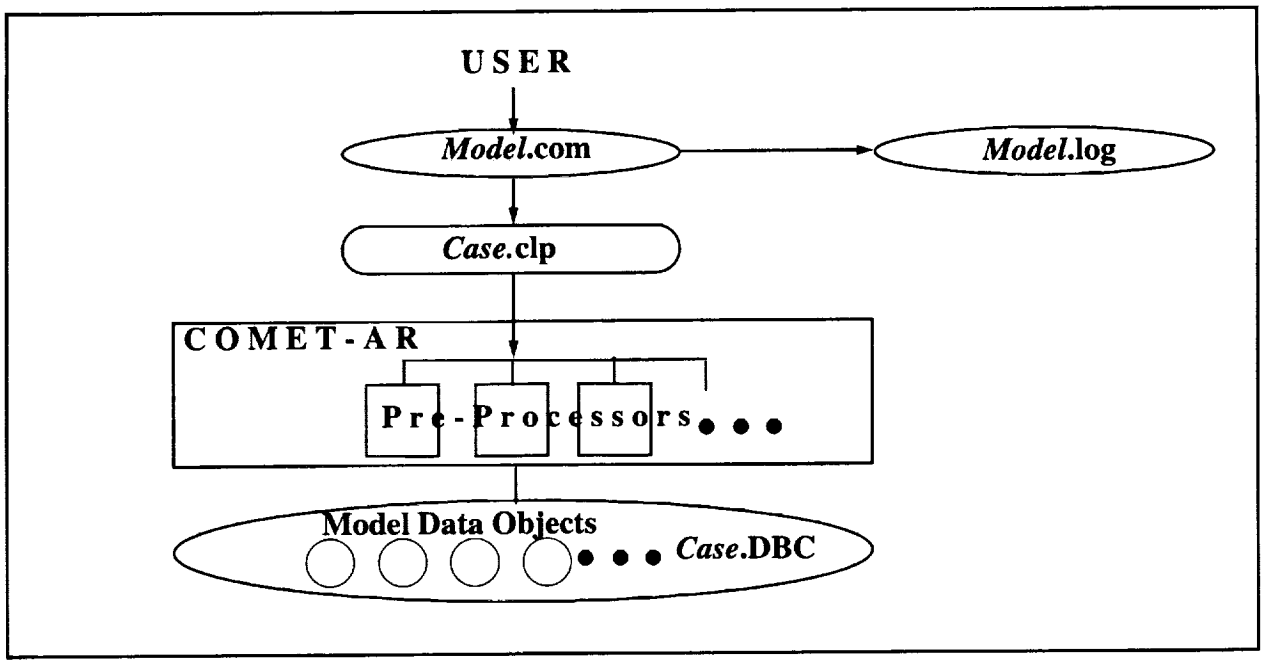


Figure 1.5-2 User Interface for Pre-Processing Without PATRAN

Two files must be created by the user: i) the *Case*.clp file, which is a COMET-AR command-language procedure file containing the commands necessary to generate a model with the various COMET-AR pre-processors (here *Case* denotes a user-selected case name); and ii) the *Model*.com file, a UNIX script file that runs the COMET-AR start-up processor (also called the COMET-AR macro-processor) which in turn invokes the *Case*.clp file. The result of executing the *Model*.com file is that a COMET-AR database (*Case*.DBC file) is generated, as well as an optional *Model*.log file containing a printed record of the COMET-AR execution. (The first part of the filename, *Model*, is an arbitrary user-defined name.)

The steps involved in pre-processing without PATRAN are summarized next. Refer to the COMET-AR User's Tutorial for detailed examples.

Step 1:

Construct a Model Definition Procedure (call it *Case*.clp) to generate the initial COMET-AR finite element model. Instructions are given in the chapter on Model Definition Procedures, in Part II of this manual. Basic model definition procedures have input arguments and contain a simple list of processor commands driving various COMET-AR pre-processors, described in Part III. More sophisticated model definition procedures involving looping and conditional statements and variables (called macrosymbols) can be constructed by referring to a separate manual on "CLIP Procedure Directives" [2] or consulting the COMET-AR Tutorial [1] for examples.

Step 2:

Construct a UNIX script file (call it *Model*.com) to initiate COMET-AR execution and invoke the model definition procedure created in Step 1. The form of the *Model*.com file is as follows.

Sample *Model*.com File

```
comet-ar
*open 1, Case.DBC
*add Case.clp
*call Case ( ... input arguments ... )
*stop
```

The "comet-ar" line executes the COMET-AR start-up/control processor. The *open directive creates a new database file called *Case*.DBC to store the model. Then the *add directive compiles the user-written *Case*.clp procedure file, and the *call directive invokes it, causing the model to be generated in the database file called *Case*.DBC, where *Case* represents a user-defined name for the case (i.e., problem) being analyzed. There may be other input arguments to procedure *Case*; depending on how the user has written it (see chapter on Model Definition Procedures in Part II for details). Finally, the *stop directive terminates the COMET-AR execution, making sure that the database is properly closed.

Step 3:

Execute the *Model*.com file as you would any UNIX script file and save the printed output in a *Model*.log file, e.g., using the following UNIX command:

Model.com >& *Model*.log &

which would cause the COMET-AR to run in batch (background) mode. After the run has completed successfully, proceed to either the Solution or Post-Processing phase.

1.5.3.2 Pre-Processing With PATRAN

When using PATRAN to define the initial finite element model, the user still must perform the last two steps of the "Pre-Processing Without PATRAN" recipe (see previous subsection), but two new initial steps are necessary: i) PATRAN model definition, and ii) conversion of the PATRAN database (i.e., Neutral File) to a COMET-AR model definition procedure. The procedure is illustrated in Figure 1.5-3.

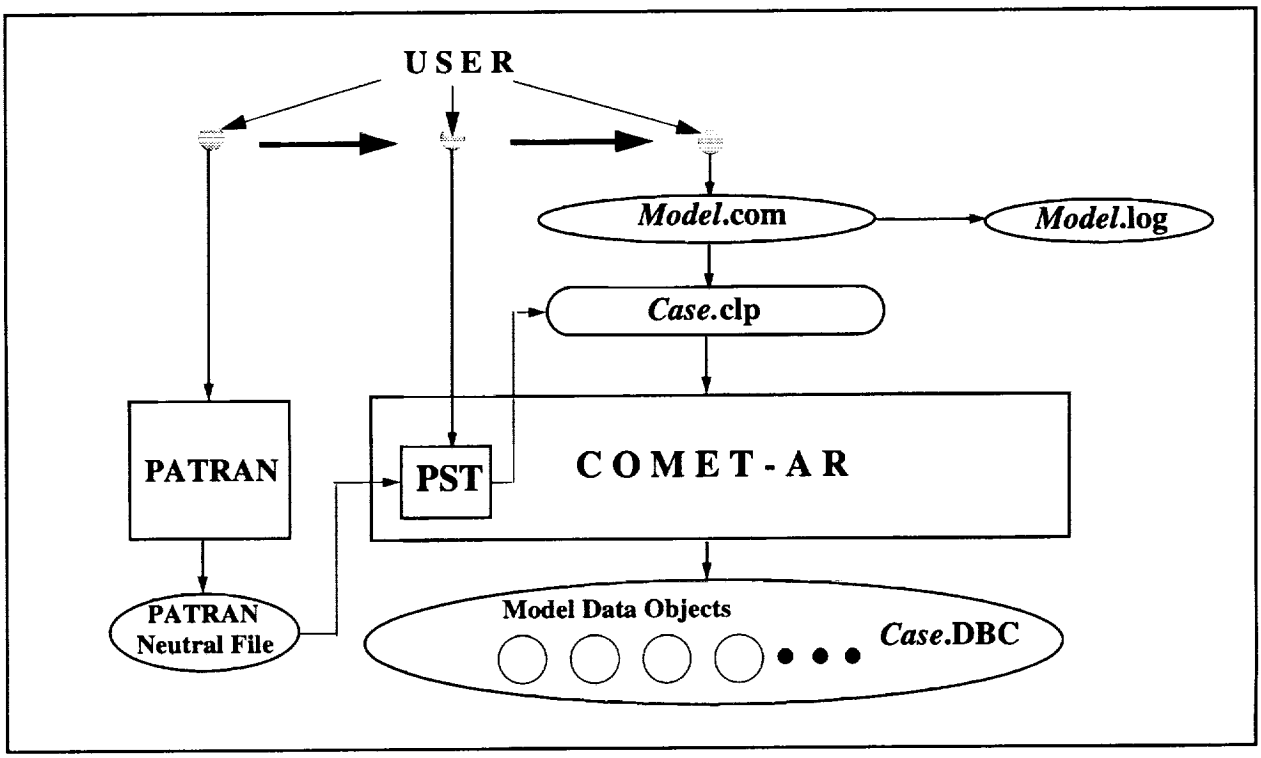


Figure 1.5-3 User Interface for Pre-Processing With PATRAN

The steps involved in generating a COMET-AR model via PATRAN are outlined in the following paragraphs.

> The PATRAN mode of pre-processing COMET-AR has only been implemented recently, and may not be quite as robust as the intrinsic form of COMET-AR pre-processing, which doesn't involve PATRAN. COMET-AR's intrinsic (non-PATRAN) pre-processing capabilities, however, are not adequate for generating complex models, as they place too much of a burden on the user. A third alternative is for the user to employ his/her favorite finite element pre-processor and write a customized data-converter, producing as output a COMET-AR Model Definition procedure file, the ingredients for which are described in Part II.

Step 1:

Construct a PATRAN model of the new problem, including all finite element information: nodes, elements/mesh, loads, boundary conditions, material properties, etc. Instructions for PATRAN are beyond the scope of this manual. Here we assume that the prospective COMET-AR user is an experienced PATRAN user (otherwise, refer to the "Post-Processing Without PATRAN" instructions). The result of the PATRAN run should be a PATRAN Neutral File containing a complete description of the initial finite element model.

Step 2:

Run the PATRAN to COMET-AR conversion processor PST to automatically generate a COMET-AR Model Definition Procedure file, called *Case*.clp. (This is analogous to Step 1 in the "Pre-Processing Without PATRAN" instructions, where the *Case*.clp file was written by the user.) Instructions for running PST in pre-processing mode are given in the section on Processor PST under the Pre-Processors chapter in Part III of this manual. Some editing of the *Case*.clp file may be required by the user to do such things as selecting the COMET-AR element type name, and defining material and fabrication (i.e., section) properties.

Step 3:

Continue by performing Steps 2 and 3 of the "Pre-Processing without PATRAN" instructions, in which a *Model*.com file is written to invoke the *Case*.clp file, and the former file is executed.

1.5.4 Solution Phase

After the model has been successfully defined (see Pre-Processing Phase), the Solution Phase can begin. During the Solution Phase, the user invokes one of COMET-AR's standard Solution Procedures, and an analysis is performed that produces various structural response data in the database. If adaptive mesh refinement has been selected by the user (currently possible only with linear static analysis), a series of solutions and corresponding updated meshes will be produced, and all related data (throughout the mesh-update history) will also be available in the database. The procedure is illustrated in Figure 1.5-4.

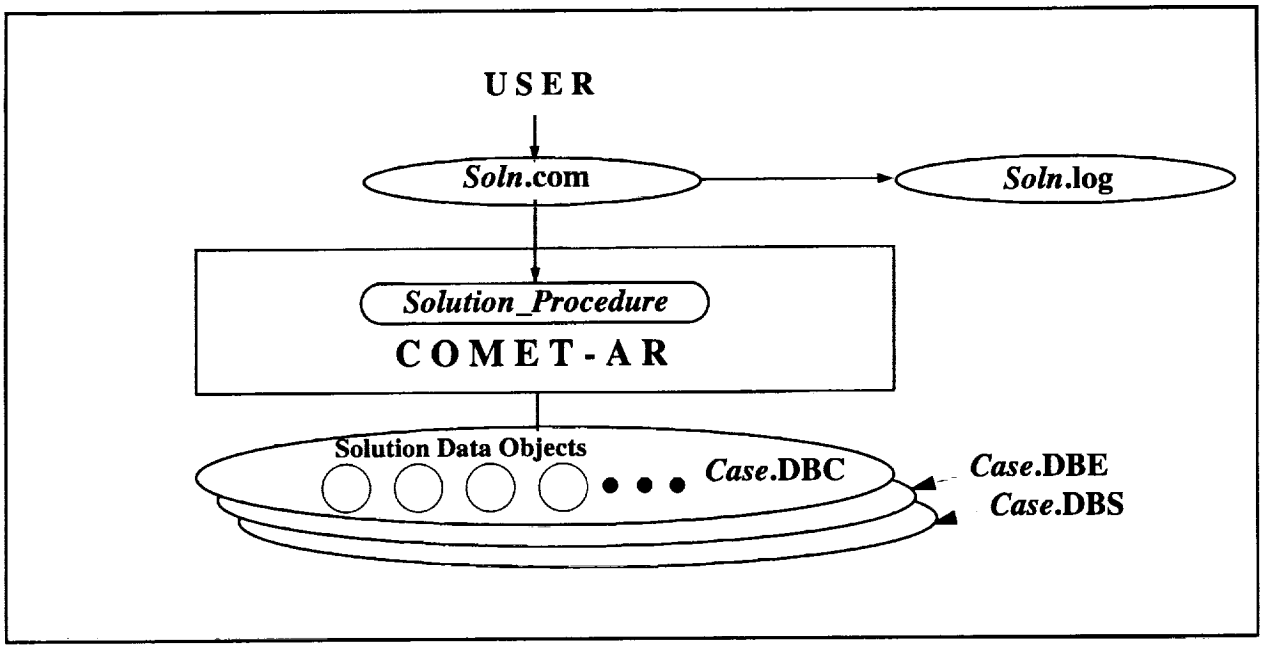


Figure 1.5-4 User Interface During the Solution Phase

The user interacts with COMET-AR by writing a UNIX script file, arbitrarily called *Soln*.com, which invokes the desired COMET-AR standard solution procedure. If the solution procedure is a basic (non-adaptive) one such as L_STATIC_1 or NL_STATIC_1, the output mesh will be the same as the input mesh. If an adaptive solution procedure such as AR_CONTROL is selected, a number of new, adaptively refined meshes and corresponding solutions will reside on the database. The *Case*.DBC file will contain most of this data. The *Case*.DBE and *Case*.DBS files will optionally contain the latest version of the element and system stiffness matrices, respectively, which can be discarded immediately if disk space is a problem.

The steps involved in performing a solution with COMET-AR are summarized next. The main requirement for the user is to become familiar with using the various COMET-AR Solution Procedures described in Part II of this manual, so that a reasonable choice can be made for both the procedure type and its associated input arguments.

Step 1:

First duplicate the COMET-AR database file, *Case*.DBC, generated during pre-processing, renaming one of the copies to *Case*_model.DBC. This is just a precaution in case you decide to repeat the solution from scratch, in which case you will probably want a fresh database file (with no extraneous solution data) without having to re-generate the model as well. The *Case*_model.DBC file provides a backup for this purpose.

Step 2:

Construct a UNIX script file (call it *Soln*.com) to initiate COMET-AR execution and invoke the solution procedure of your choice. The form of the *Soln*.com file is as follows.

Sample "*Soln*.com" File

```
comet-ar
*open/rold 10, proclib.gal
*set plib = 10
*open 1, Case.DBC   . (this line is not always required)
*call Solution_Procedure ( ... input arguments ... )
*stop
```

The "comet-ar" line executes the COMET-AR start-up/control processor. The first "*open" directive opens the standard COMET-AR procedure database file (proclib.gal), which contains all of COMET-AR's solution and utility procedure files in compiled form. The "*set plib" directive tells COMET-AR where to look for these procedures. The second "*open" directive opens the COMET-AR database file containing the model definition, which was just created in the pre-processing phase. This file may contain solution data too if the current run is a re-start, or continuation. (Some solution procedures open the .DBC file internally and so the second *open directive may not be required in the *Soln*.com file). Next, the "*call" directive invokes the user-selected solution procedure to perform an analysis with COMET-AR. The names and input-argument options for the various COMET-AR solution procedures are described in Part II of this manual. Finally, the *stop directive terminates the COMET-AR execution, making sure that the database is properly closed.

Step 3:

Execute the *Soln*.com file and save the printed output in a *Soln*.log file, e.g.,

Soln.com >& *Soln*.log &

which would cause the COMET-AR to run in batch (background) mode. After the run has completed successfully, proceed to the Post-Processing phase (and/or repeat/re-start the solution).

1.5.5 Post-Processing Phase — Result/Model Evaluation

1.5.5.1 Post-Processing Without PATRAN

Several COMET-AR processors are available for post-processing solution and/or model data (see Figure 1.5-5). The most powerful is processor ARGx, which is an interactive-graphics color display processor that may be used to visualize the model and solution in various ways, unlike most of the other processors in COMET-AR. Unlike most of the other processors in COMET-AR, ARGx is driven by a graphical user interface (GUI) and is typically used in stand-alone mode. ARGx is particularly useful for verifying and visualizing the models and solutions generated during adaptive mesh refinement.

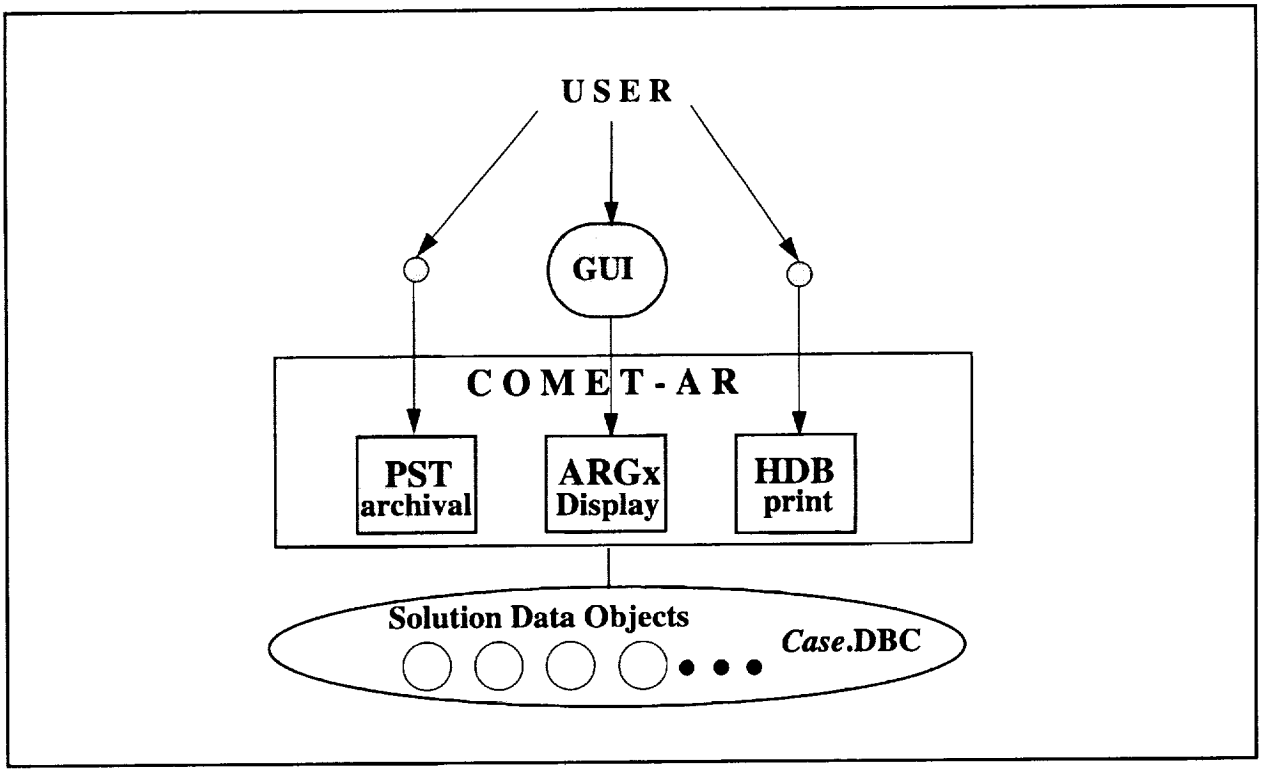


Figure 1.5-5 User Interface(s) During Post-Processing Phase Without PATRAN

Two other COMET-AR processors can be valuable for post-processing: processor HDBprt, which allows the user to print selected parts of the database; and processor PST, which allows the user to find and archive critical solution data such as maximum stresses, and stresses at prescribed locations in the model, designated either by coordinates, closest node number, or closest element number. (Processor PST is also the COMET-AR/PATRAN translator.)

The steps for post-processing without PATRAN are summarized as follows. These steps may be taken in any order.

Step 1:

Execute the COMET-AR interactive-graphics post-processor, ARGx. This processor allows you to look at the deformed geometry, color contours of solution quantities, and to verify load directions and nodal boundary conditions as well. It also has some graphing (x-y plot) capabilities and will display numerical values at locations indicated by mouse selection. (See the section on Processor ARGx in the chapter on Post-Processors in Part III of this manual.)

Step 2:

Execute COMET-AR post-processor HDBprt to get list-type printed displays of selected node and/or element data. In fact, use HDBprt to examine any data objects of interest. (See the section on Processor HDBprt under Post-Processors chapter in Part III of this manual.)

Step 3:

Execute Processor PST to archive selected quantities such as the stress at a prescribed location or node, maximum stress, etc. The selected values are placed in the database for subsequent post-processing by the user. (See the section on Processor PST under Post-Processors chapter in Part III of this manual.)

1.5.5.2 Post-Processing With PATRAN

PATRAN may be used to post-process a COMET-AR model/solution whether or not PATRAN was used to generate the initial finite element model. Even if the initial finite element model was generated with PATRAN, if COMET-AR adaptive mesh refinement is employed to perform the solution, a new finite element model will be part of COMET-AR's output, and will have to be translated to PATRAN as well. The situation is illustrated in Figure 1-10.

COMET-AR processor PST is used first to generate PATRAN Results and Neutral files from the COMET-AR database. Then, the user may interact directly with PATRAN with its own native user interface. The steps needed to perform this procedure follow Figure 1.5-6.

The COMET-AR to PATRAN interface processor PST is a recent addition to COMET-AR and it may not be as robust as some of the of the processors. Adamant PATRAN users may, therefore, need to contact the development team via NASA for assistance. Others may find that the COMET-AR processor ARGx provides most of the necessary graphical display functions (and more) that are provided by PATRAN. (See the previous subsection for a discussion of ARGx).

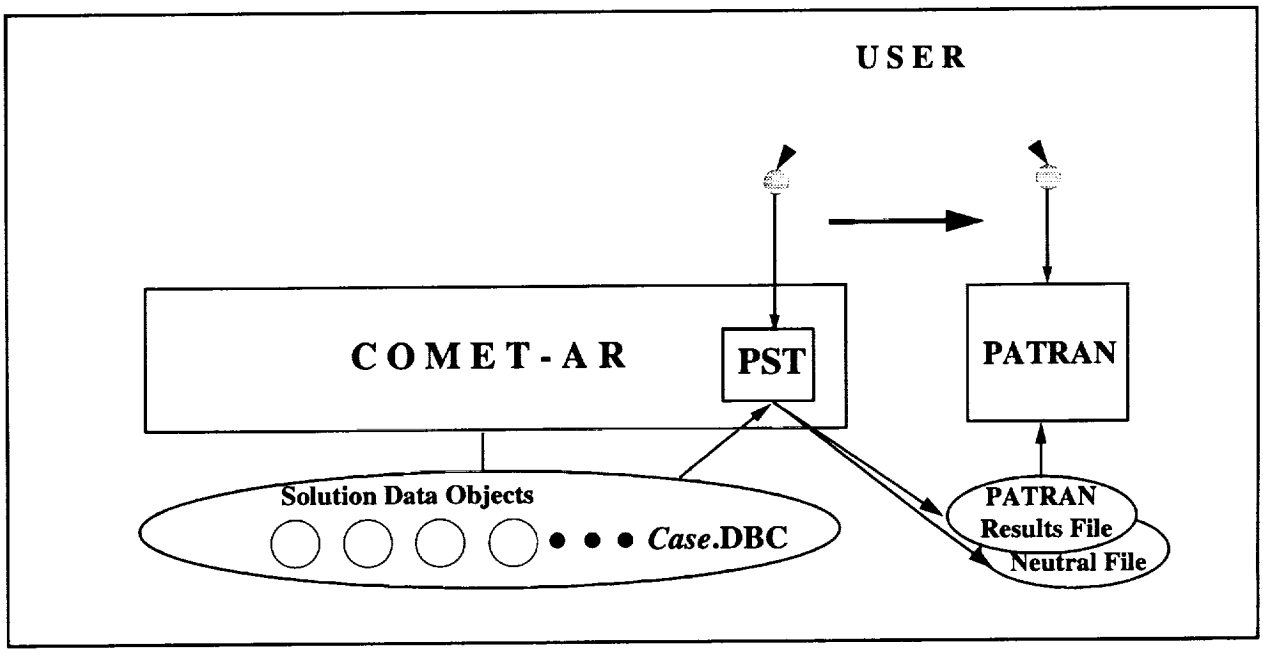


Figure 1.5-6 User Interface During Post-Processing Phase With PATRAN

Step 1:

Execute the COMET-AR processor PST in COMET-AR_to_PATRAN mode. This will translate both model and solution data from the COMET-AR database to the PATRAN Neutral and Result files. If adaptive mesh refinement is being used (e.g., via solution procedure AR_CONTROL), the finite element model will be changing as well as the solution. In this case, the original PATRAN Neutral File will no longer be valid and a new one consistent with the current solution will have to be generated via PST. COMET-AR saves both model and solution data for all intermediate meshes generated during adaptive refinement, and the user may translate any of these models/solutions to PATRAN for post-processing with processor PST. (Refer to the section on Processor PST under the Post-Processors chapter in Part III of this manual for usage details.)

Step 2:

Execute PATRAN and display the model, results, etc. This step will depend on the experience-level of the PATRAN user. It is not covered in the COMET-AR manual.

Step 3:

The user can always employ the COMET-AR post-processors described in the preceding subsection in addition to PATRAN. Different COMET-AR users may prefer to use different post-processors to display results for the same analysis.

1.6 How to Use This User's Manual

The COMET-AR User's Manual is partitioned into five parts, as shown in Table 1.6-1.

Table 1.6-1 Organization of the COMET-AR User's Manual

Part	Title	Contents
I	Introduction	Overview of COMET-AR and how to use it.
II	Procedures	Describes COMET-AR command-language procedures, including user-written Model Definition Procedures, pre-defined Solution Procedures, and subordinate Utility Procedures. There is a separate section here for each procedure.
III	Processors	Describes COMET-AR FORTRAN processors, including pre-processors, element processors, constitutive processor, matrix/vector processors, post-processors, and special-purpose processors. There is a separate section for each processor.
IV	Database	Describes the COMET-AR database, how it is partitioned into data files and data objects, and how each data object is partitioned into attributes. Also explains how the database evolves during analyses with adaptive mesh refinement.
V	Solid-Model Interface	Describes two options the user has for defining the underlying geometry of a model in conjunction with adaptive mesh refinement: i) the discrete solid-model description, based on the initial finite element model, and ii) the continuous solid-model description, which is more accurate but requires a number of user-written subroutines that are cumbersome for complex structures.

The correspondence between these parts of this manual and the three phases of a COMET-AR analysis is shown in Table 1.6-2, which indicates where to look during each phase.

Table 1.6-2 Correspondence Between Documentation and Analysis Phase

Phase	Where to Look in this User's Manual
1) PRE-PROCESSING	Consult Part II, under Model Definition Procedures chapter; or if using a PATRAN model, consult Processor PST in Part III.
2) SOLUTION	Consult Part II, under Basic Solution Procedures chapter, and/or Adaptive Solution Procedures (for adaptive mesh refinement).
3) POST-PROCESSING	Consult Part II, under Post-Processors chapter; in particular, see sections on processors ARGx, HDBprt, and PST.

A "road map" for performing COMET-AR analysis in conjunction with the documentation in the present User's Manual is provided in Figure 1.6-1.

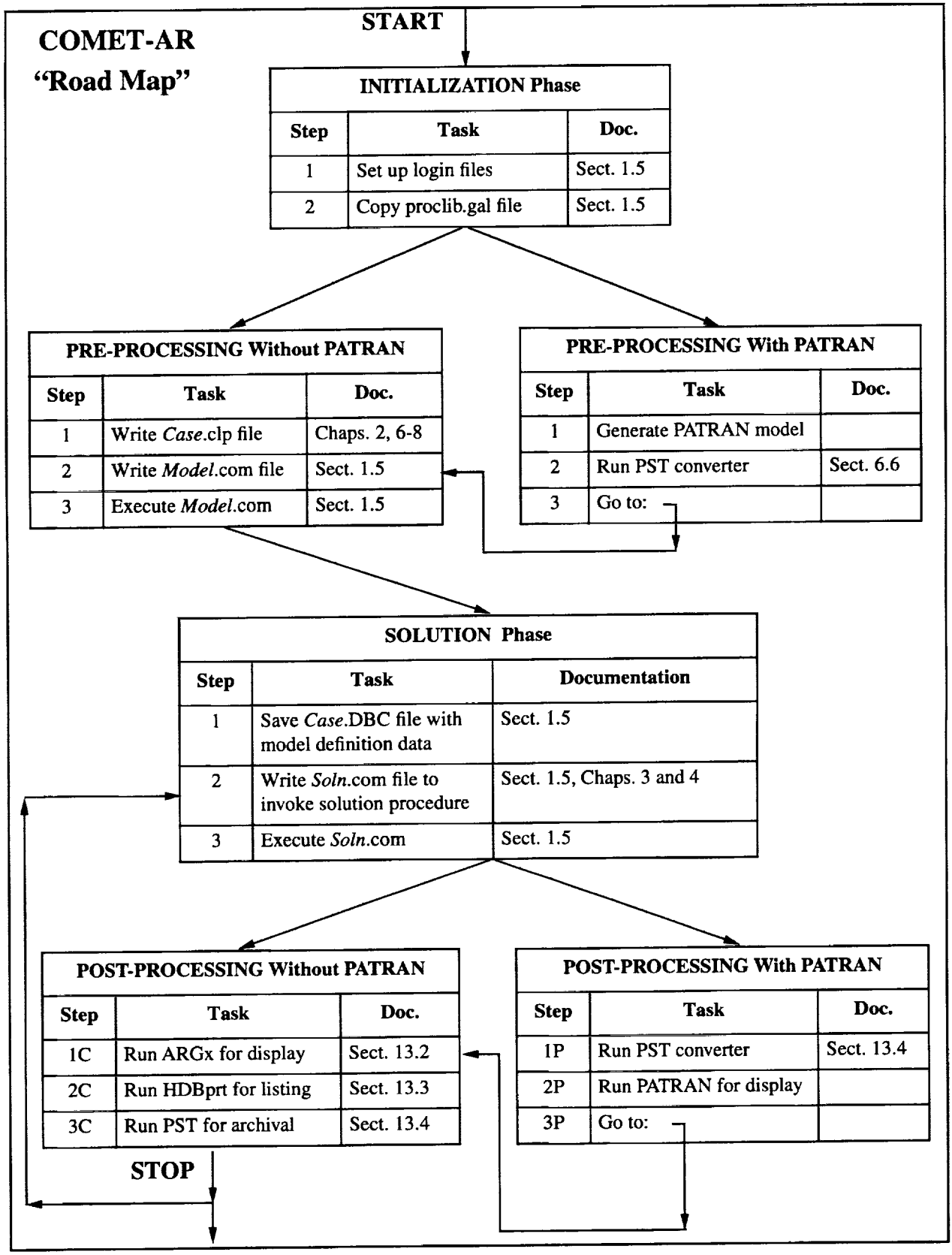


Figure 1.6-1 "Road Map" of COMET-AR Use

1.7 Related COMET-AR Documentation

Table 1.7-1 provides a summary of recommended supplementary documentation to the COMET-AR User's Manual.

Table 1.7-1 Summary of Related COMET-AR Documentation

Document	Ref.	Contents
COMET-AR Tutorial	[1]	Walk-through examples of using COMET-AR for various kinds of analysis; recommended for beginners.
COMET-AR HDB Manual	[3]	Detailed description of high-level database access; recommended for software developers.
COMET-AR DB Manual	[5]	Detailed description of generic database utilities employed by HDB; recommended for software developers.
CSM Generic Element Processor Manual	[8]	Contains instructions for adding new element types (i.e., processors) to COMET-AR; recommended for element developers.
CSM Generic Constitutive Processor Manual	[9]	Contains instructions for adding new constitutive models to COMET-AR; recommended for constitutive model developers.
CLIP Manuals	[2]	Detailed description of command/procedure language employed by COMET-AR.
GAL Manual	[4]	Detailed description of file-management utilities employed by HDB (via DB); recommended for software developers.
COMET User's Manual	[6]	Counterpart to this manual for the COMET code, which is an ancestor of COMET-AR; however, does not cover command-language procedures or database.
COMET Procedure Manual	[7]	Describes command-language (CLIP) procedures available in the COMET code from which COMET-AR was derived.

1.8 Command Language Summary

As described in Section 1.5, COMET-AR is controlled by the user via a command language, called CLAMP (Command Language for Applied Mechanics Processors), which is processed by a software architectural component called CLIP (Command Language Interface Program). Command input begins immediately after the COMET-AR macro-processor is first executed on the user's host operating system (i.e., by entering the macro-processor name, "comet-ar," in a UNIX script file). At that point, you have access to a variety of commands which fall into two classes.

1) **CLIP Directives**: These are generic COMET-AR commands that begin with an asterisk (*), such as *OPEN, *CALL, *PROCEDURE, and *ADD; and perform global control functions, such as opening a database file, calling a command-language procedure file, creating a command-language procedure, and directing input from another file (or compiling a procedure). More advanced CLIP directives form the basis of many standard command-language procedures (see Section 1.4) and include such things as macro-symbol variable definitions, looping directives and conditional statements. Users may have to become familiar with the more advanced features if they are either: i) writing complex Model Definition Procedures; or ii) participating in the development of COMET-AR by writing additional Solution Procedures. CLIP directives may be entered while executing any COMET-AR processor.

2) **Processor Commands**: These are commands that are specific (i.e., local) to each of COMET-AR's independently executable processors (see Section 1.4). One especially important command is the RUN command, which is processed by the COMET-AR macro-processor, and is used to run other processors. Once another processor's execution has been initiated via the RUN command, the user (or procedure writer) may enter only: i) commands that are recognized by that particular processor; or ii) CLIP directives, which are recognized in all COMET-AR processors (by the underlying CLIP architectural utilities that respond to them).

Processor commands for each of COMET-AR's processors are described in corresponding sections of Part III. CLIP directives, which have the same description for all of the procedures appearing in Part II, and commands that are common to all COMET-AR processors, are summarized in the following subsections.

> See the CLIP Manual [2] for a comprehensive description of the CLAMP language, including directives (Vol. II), command syntax (Vol. I), and the FORTRAN interface to this language for processor developers (Vol. III). An intermediate description, somewhat more expanded than presented here but less detailed than in [2], may be found in reference [6].

1.8.1 CLIP Directives

CLIP directives are special commands that are understood and processed by the COMET-AR architectural utility CLIP, and are not interpreted by individual processors. (A directive is to CLIP like an ordinary command is to a processor.) Directives may appear in all forms of COMET-AR input, but some directives, such as the *PROCEDURE directive and nonsequential processing directives, must be used only within command language procedures (called CLIP procedures).

A directive is distinguished from an ordinary command by beginning with a keyword prefixed by an asterisk (*). The keyword (verb) may be followed by a verb modifier, qualifiers, and/or parameters, as required by the syntax of the particular directive. A brief description of the most important directives is given here. For a more complete description, consult Vol. II of reference [2].

The CLIP directives are grouped in Table 1.8-1 by function; detailed descriptions of the directives are contained in the following subsections.

Table 1.8-1 Summary of CLIP Directives

Directive	Function
Database Directives	
*OPEN	Opens a COMET-AR data file (also called a "library").
*CLOSE	Closes a COMET-AR data library.
*TOC	Prints a table of contents of a data library (listing datasets).
*RAT	Prints a table of contents of records in a dataset (record access table).
*PRINT	Prints contents of a dataset within a data library. (It is often more convenient and meaningful to employ the PRINT command in "post-processor" HDBprt for object-oriented datasets.)
*COPY	Copies datasets or dataset records within or across data libraries.
*DELETE *ENABLE	Deletes (i.e., disables) datasets or records within a data library. Enables previously deleted (i.e., disabled) datasets or records.
*FIND	Returns information on datasets or records.
*RENAME	Renames datasets or records.
Procedure Management Directives	
*SET PLIB	Sets procedure library index as source of command procedures.
*PROCEDURE *END	Initiates definition of a command procedure. Terminates definition of a command procedure.
*CALL	Invokes a command procedure with optional argument replacements.
Non-Sequential Processing Directives (in Procedures Only)	

*IF
*ELSE
*ELSEIF
*ENDIF
	Conditional branching constructs.

Table 1.8-1 Summary of CLIP Directives (Continued)

Directive	Function
*DO *ENDDO	Do-Looping constructs.
*WHILE *ENDWHILE	While-Looping constructs.
*JUMP	Transfer control to specified label.
*RETURN	Forces exit from command procedure.
MacroSymbol Directives	
*DEFINE *UNDEFINE	Defines a macrosymbol (or macrosymbol array). Deletes a macrosymbol(s).
*SHOW MACRO	Shows current definition of macrosymbol(s).
*GAL2MAC *MAC2GAL	Defines a macrosymbol from a database record. Defines a database record from a macrosymbol.
Built-in Macros	Common constants, mathematical functions, generic functions, reserved variables, Boolean functions, logical functions, string concatenation, string matchers, and status macros.
Miscellaneous Directives	
*ADD	Redirects input to come from a specified text file; compiles procedures.
*ECHO	Turns command/directive print-echo on or off.
*HELP	Lists information from a Directive HELP file.
*REMARK	Prints a remark (or comment) line.
*SET *SHOW	Sets various control parameters (e.g., output device index). Shows various control parameters.
*UNLOAD *LOAD	Unloads contents of a data library to an ASCII file. Loads contents of a data library from an "UNLOADED" ASCII file.

1.8.1.1 Database-Oriented Directives

Database-oriented procedure directives provide the user with direct access to the COMET-AR global database from within procedures and other input files. The *OPEN directive is particularly important, as it must be used to open a database file (i.e., a data library) before any COMET-AR processors can be engaged. The other directives in this subsubsection are optional. For example, the *PRINT directive is rarely used; instead the PRINT command within processor HDB is preferred for obtaining object-oriented printouts. The *TOC directive is often useful for getting an overview of the data library before using the PRINT command, and may be used interactively within HDB.

*1.8.1.1.1 The *OPEN Directive*

The *OPEN directive opens a data library. The directive format is:

*OPEN *ldi filename /qualifier*

where *ldi* is the library identification number (or "logical device index") and *filename* is the external name of the permanent library file. If *ldi* is omitted, it will default to the first free library number (which is 1 at the beginning of a COMET-AR execution). If *filename* is omitted, it will default to fort.*ldi*. Once a library has been named, it may be referenced by number (i.e., by the *ldi*), in subsequent directives such as *CLOSE, *TOC, etc.

The most commonly used qualifiers include NEW, OLD, and READ. The qualifier NEW will open a new (empty) library. The qualifier OLD will open an existing library (or print an error if the library does not exist) and the qualifier READ will open an existing library for read-only operations. If no qualifiers are used, an existing library will be open if it exists or a new one will be created. In either case, write permission is the default.

*1.8.1.1.2 The *CLOSE Directive*

The *CLOSE directive closes an open data library. The directive format is:

*CLOSE *ldi /qualifier*

where *ldi* is the library identification number, which if omitted, defaults to all active libraries. A closed library cannot be accessed again until it has been re-opened. There is only one optional qualifier, DELETE, which deletes the file upon closing. The *CLOSE directive is automatically invoked internally by COMET-AR in response to the RUN EXIT command.

*1.8.1.1.3 The *TOC Directive*

The *TOC directive prints a table of contents of datasets within a library. The directive format is:

*TOC *ldi* [*ids* | *dsname*]

where *ldi* is the library identification number. If the optional *ids* or *dsname* parameters are omitted, a table of contents of all datasets in the library is printed. A partial table of contents may be obtained by specifying either *ids*, a range of dataset sequence numbers, or *dsname*, a dataset name which may have a wild character (*) to indicate more than one match is desired. For example:

*TOC 1 1:10

will provide a table of contents information about datasets 1 through 10, while

TOC 1 NODAL.

will provide a table of contents of all nodal datasets, i.e., those whose first name is NODAL.

*1.8.1.1.4 The *RAT Directive*

The *RAT directive will print a table of contents of records within a given dataset (or datasets). This is typically referred to as a Record Access Table (or RAT). The directive format is:

*RAT *ldi* [*ids* | *dsname*]

where the directive parameters have the same meaning as for the *TOC directive.

*1.8.1.1.5 The *PRINT Directive*

The *PRINT directive prints the actual data within one or more dataset records. The directive format is:

*PRINT *ldi* {*ids* | *dsname* } *record_name* [/OUT=*unit*]

where *record_name* is the name of the record, which may specify a range of records if the dataset consists of record groups. For example, the directive

PRINT 1 NODAL.DISPLACEMENT NVT.1:10

would cause records NVT.1 through NVT.10 within dataset NODAL.DISPLACEMENT on library 1 to be printed. A more meaningful way to do this (in general) is to use the PRINT command in processor HDBprt and request that the nodal displacements for nodes 1 through 10 be printed (where nodal displacements are stored as a Nodal Vector Table, or NVT, data object (see Part IV on the COMET-AR database). The optional /OUT=*unit* qualifier enables the user to redirect the printed output to a file which will be named "fort.*unit*."

*1.8.1.1.6 The *DELETE and *ENABLE Directives*

The *DELETE directive disables a specified set of datasets from a library. The directive format is:

*DELETE *ldi* {*ids* | *dsname* }

where *ldi* is the library identification number, *ids* represents a range of dataset sequence numbers, and (alternatively) *dsname* represents a dataset name specification, with optional wild characters (*). Disabled datasets remain in the database, but may not be accessed by subsequent directives or

processors unless they are enabled. To enable a dataset(s) that has been disabled via the *DELETE directive, the *ENABLE directive may be used. It has the following format.

*ENABLE *ldi* {*ids*	*dsname* }

Disabled datasets appear in *TOC listings with an asterisk next to the sequence number. After they have been enabled, the asterisk no longer appears. When a library is copied to another library (via the *COPY directive), disabled datasets are not copied to the destination library. This provides a way of truly deleting datasets from the database by creating a new library with only active datasets. (Another way is the *PACK directive, which deletes and copies datasets in place; however, this is a rather risky directive. If it is interrupted by a system crash, the whole data library may be lost.)

*1.8.1.1.7 The *COPY Directive*

The *COPY directive copies a dataset to a new dataset, either within a single library, or across libraries. The directive format is:

*COPY *ldi _to* [*dsname_to*] = *ldi_from* {*ds_name_from*	*ids_from* }

where *ldi_to* is the destination library number, *dsname_to* is the optional destination dataset name (which defaults to the source dataset name(s)), *ldi_from* is the source library number, and *ds_name_from* (or alternatively *ids_from*) is the source dataset name (or sequence number range) specification. For example:

```
COPY 2 = 1
```

would copy all datasets from library 1 to library 2;

```
COPY 2 = 1  NODAL.*
```

would copy all datasets with first name NODAL from library 1 to library 2; and

```
COPY 1  NODAL.VELOCITY  =  1  NODAL.DISPLACEMENT
```

would copy the contents of the NODAL.DISPLACEMENT dataset to a new dataset called NODAL.VELOCITY, both within library 1.

1.8.1.2 Procedure Management Directives

Procedure management directives provide a means of defining and invoking COMET-AR command-language procedures, which may contain a mixture of other directives and processor commands, constituting a functional unit, that may be parametrized via procedure arguments.

*1.8.1.2.1 The *PROCEDURE and *END Directives*

The *PROCEDURE directive initiates the definition of a procedure. The directive format is:

*PROCEDURE *procedure_name* (*argument_list*)

where *procedure_name* is the name of the procedure. If there is an argument list (*argument_list*) the parentheses are mandatory, and there must be a space separating the procedure name and the first parenthesis. The argument list may contain up to 100 formal arguments in the form:

(*arg1* = *default1* ; *arg2* = *default2* ; . . .)

where *arg1* and *arg2* represent argument names, and *default1* and *default2* represent their default values. Default values for arguments are optional (i.e., the =default phrases are optional). Non-default values for arguments are provided at "run" time, via the *CALL directive. Within procedures, argument names enclosed in square brackets, i.e.,

[*arg_name*]

are replaced by the assigned or default symbolic values given them via the *CALL or *PROCEDURE directives.

Finally, the *END directive is used to terminate a procedure definition (i.e., all procedures must begin with a *PROCEDURE directive and end with the directive).

*END

*1.8.1.2.2 The *CALL Directive*

The *CALL directive invokes a COMET-AR procedure. The directive format is:

*CALL *procedure_name* (*argument_list*)

where *procedure_name* is the name of the procedure and, as in the *PROCEDURE directive, there must be at least one space separating the procedure name and the first parenthesis before the argument list. When calling a procedure, the argument list takes the form:

(*arg1* = *value1* ; *arg2* = *value2* ; . . .)

where *arg1* and *arg2* are argument names (which must also appear in the corresponding *PROCEDURE directive) and *value1* and *value2* are their user-specified values (or strings). The order in which the arguments appear in the *CALL directive is arbitrary. Not all procedure arguments need be explicitly mentioned, in which case they will take on their default values (see description of the *PROCEDURE directive). Procedures must first be compiled via the *ADD directive (see Miscellaneous Directives) before they can be called.

*1.8.1.2.3 The *SET PLIB Directive*

The *SET PLIB (Set Procedure Library) directive associates callable procedures with a data library, indicating that all subsequent procedure calls (via the *CALL directive) will access compiled procedures resident on a particular data library. It also indicates that all subsequent procedure compilations, via the *ADD directive, will produce compiled procedures that are to be stored in the specified data library. The directive format is:

```
*SET PLIB  ldi  [dsname ]
```

where *ldi* is the library identification number, and *dsname* is the name of the dataset in which callable procedures are assumed to reside. If *ldi* is omitted, it defaults to zero, which means that procedures reside on ordinary ASCII disk files. If *dsname* is omitted, the default dataset name: CALLABLE.PROCEDURES will be assumed. In the absence of the *SET PLIB directive, all callable procedures are assumed to exist as separate ASCII files within the current disk directory.

1.8.1.3 Non-Sequential Processing Directives

Some of the most useful directives are those that provide the means for nonsequential command and directive processing. The directives in this category may only be used within a procedure.

*1.8.1.3.1 The *IF, *ELSEIF, *ELSE, and *ENDIF Directives (Conditional Branching)*

This construct is also known as the BLOCK IF directive. The format is:

```
*IF < logical expression > /THEN
:
*ELSEIF <logical expression > /THEN
:
additional ELSEIF's
:
*ELSE
:
*ENDIF
```

This construct behaves much like the FORTRAN if-then-else construct. Both the *ELSEIF and the *ELSE may be omitted. The logical expression must evaluate to either <TRUE> or <FALSE> (see Macrosymbol Directives) and is typically of the form:

< a relational_qualifier b >

where *a* and *b* may be either macrosymbols, numbers, or logical expressions and the *relational_qualifier* may be any one of those listed in Table 1.8-2.

Table 1.8-2 Relational Qualifiers in Logical Expressions

Expression	Evaluates to:
a /EQ *b*	<TRUE> if $a = b$, else <FALSE>
a /LE *b*	<TRUE> if $a \leq b$, else <FALSE>
a /LT *b*	<TRUE> if $a < b$, else <FALSE>
a /GE *b*	<TRUE> if $a \geq b$, else <FALSE>
a /GT *b*	<TRUE> if $a > b$, else <FALSE>
a /NE *b*	<TRUE> if $a \neq b$, else <FALSE>
e_1 /AND e_2	<TRUE> if both e_1 and e_2 are <TRUE>, else <FALSE>
e_1 /OR e_2	<TRUE> if either e_1 or e_2 are <TRUE>, else <FALSE>

The following is an example of a valid BLOCK IF directive construct.

```
:
*IF  < <mac1> /eq 2 > /THEN
*DEFINE/i mflag = <TRUE>
*ELSEIF  <value>  /THEN
*DEFINE/i vflag = <TRUE>
*ENDIF
:
```

*1.8.1.3.2 The *DO/*ENDDO Directives (DO-Loops)*

This set of directives provides a FORTRAN-like looping construct. The format is:

```
*DO $macro_name = i1, i2  [ , i3 ]
:
*ENDDO
```

where *$macro_name* is the name of a special type of macrosymbol (see Macrosymbol Directives) which must, as indicated, start with a *$* sign. The integers *i1* and *i2* specify the initial and final values for the loop variable, *$macro_name*. The integer *i3* specifies the increment of the loop variable. If *i3* is not given, a value of +1 will be assumed, provided $i1 < i2$. If $i1 > i2$ a value of –1 is assumed for *i3*. Examples of valid *DO loops include:

```
*DO  $i = 0, 100, 10
:
*ENDDO
```

which will cause the enclosed commands/directives to be executed 10 times, with the macrosymbol, *$i*, incremented by 10 each time; or:

```
*DO $i = 1, 100
:
*ENDDO
```

which will cause the enclosed commands/directives to be executed 100 times, with the macrosymbol, *$i*, incremented by one each time; or:

```
*DO $i = 100, 1
:
*ENDDO
```

which will cause the enclosed commands/directives to be executed 100 times, with the macrosymbol, *$i*, decremented by one each time.

There is also an alternative form of the *DO loop which uses a label to close the loop. It has the format:

```
*DO :label  $macroname  = i1, i2  [,i3 ]
:
:label
```

where *label* is the label name.

1.8.1.4 Macrosymbol Directives

Macrosymbols are variables that may be used both within COMET-AR procedures and in ordinary (non-procedural) COMET-AR input to processors. Macrosymbols may be defined with the *DEFINE directive, and deleted via the *UNDEFINE directive. A macrosymbol is decoded into an actual numerical value or string by enclosing the macrosymbol name in angle brackets, i.e.,

< macro_name >

where *macro_name* is the name of the macrosymbol, would be decoded to:

macro_value

where *macro_value* is a numerical value or alphanumeric string, depending on the type of the macrosymbol. Macrosymbol arrays are macrosymbols with numeric indices. When defining an element of a macrosymbol array, the index follows the macrosymbol name, surrounded by square brackets. For example, the phrase:

< *macro_name* [*i*] >

would decode to the value of the *i*th element of the macrosymbol array with name *macro_name*.

The *DEFINE and *UNDEFINE directives are described below, in addition to the *GAL2MAC and *MAC2GAL directives, which transfer data values between macrosymbols and the database. Also included in this subsubsection is a brief description of the rules for performing macrosymbol arithmetic and a summary of some useful built-in COMET-AR macrosymbols and functions.

*1.8.1.4.1 The *DEFINE and *UNDEFINE Directives*

The *DEFINE directive is used to define a macrosymbol or macrosymbol array. The directive format is:

*DEFINE [*/type*] *macro_name* { = | == } *definition_text*

where *DEFINE may be abbreviated as *DEF.

The *macro_name* may contain up to 12 characters. In the case of a macrosymbol array, the index and enclosing brackets are considered part of the name. The first character of a macrosymbol name must be either a letter or a dollar sign. If the latter, the second character must be a letter.

The *macro_name* and *definition_text* must be separated either by an equal sign (=) or by a double-equal sign (==). The latter is used to force global scope (i.e., to define the macrosymbol as a global macrosymbol that has meaning at all procedure levels). Permissible macrosymbol types are listed in Table 1.8-3.

Table 1.8-3 Macrosymbol Type Identifiers

Type	Meaning
A	Unprotected character string
D[*w.d*]	Double-floating-point
E[*w.d*]	Single-floating-point, engineering (exponential) notation
F[*w.d*]	Single-floating point, decimal (non-exponential) notation
G[*w.d*]	Single-floating point, engrg/decimal notation as needed
I	Integer
N	Nearest integer
P	Protected character string

The D, E, F, and G types are analogous to the field specifications appearing in FORTRAN FORMAT statements, as are the optional *w* (width) and *d* (decimal) specifications. All macrosymbols

decode to character strings when enclosed in angle brackets (< >); hence, numeric macrosymbols may be used to construct character strings by concatenation with other character strings by decoding them (e.g., A<*i*> would decode to A1 if the value of the macrosymbol $i = 1$).

Only 731 user-defined macrosymbols may be active at any one time. This restriction applies to macrosymbol arrays as well. Each time of the array is considered to be one macrosymbol. One cannot then define an array of length 732. To overcome this restriction, the user may wish to purge macrosymbols that are not longer needed. This is done via the *UNDEFINE directive, which has the format:

*UNDEFINE [/GLOBAL] *macro_name_list*

where *macro_name_list* is a list of macrosymbols to be undefined, and the optional GLOBAL qualifier will delete all macrosymbols of the specified name(s) at all procedure levels, up to the highest (global) level. If the GLOBAL qualifier is omitted, macrosymbols above the current procedural level will not be deleted.

*1.8.1.4.2 The *SHOW MACROS Directives*

The *SHOW MACROS directive is used to print the current values of macrosymbols via:

*SHOW MACROS [*macro_name(s)* /BI]

where *macro_name*(s) designates the names; the /BI qualifier denotes built-in macrosymbols. If no macrosymbol names are specified, all user-defined macrosymbols are printed by default.

*1.8.1.4.3 The *GAL2MAC and *MAC2GAL Directives*

This pair of directives provides the user with a means of creating a macrosymbol from a global dataset record (*GAL2MAC) or creating a global dataset record from a macrosymbol (*MAC2GAL). These directives may be abbreviated as *G2M and *M2G, respectively. The directive formats are:

*G2M *record_id* /NAME=*macro_name* /TYPE=*macro_type* /M=*n_items* /IOFF=*offset*

and

*M2G *record_id* /NAME=*macro_name* /TYPE=*macro_type* /M=*m_items* /IOFF=*offset*

where *macro_name* is the input (*M2G) or output (*G2M) macrosymbol name, *n_items* is the number of items to be read into (*M2G) or written into (*G2M), and *offset* is the integer offset from the beginning of the dataset record being written to (*M2G) or copied from (*G2M). In either case, the *macro_type* refers to the data type of the resulting entity (i.e., *G2M requires the

macrosymbol type and *M2G requires the record type), and *m_items* refers to the maximum number of items to be transferred. Regarding defaults, *m_items* defaults to 100 if the /M qualifier is not specified, *offset* defaults to 0 if the /IOFF qualifier is not specified, and *macro_type* defaults to the macrosymbol (*M2G) or dataset record (*G2M) data type.

The *record_id* consists of three items separated by either commas or spaces. There are two permissible forms for *record_id*:

ldi ds_name record_name

or

ldi ids record_name

where *ldi* is the library identification number, *ds_name* is the dataset name and *ids* is the dataset sequence number.

The parameter *record_name* may assume several forms. It may be omitted (only with the *G2M directive), in which case a macrosymbol array containing each item of every record will be created. If there is only a single item to be read then the macrosymbol array becomes a simple (unsubscripted) macrosymbol. The *record_name* may also consist of a KEY and a CYCLE or CYLES. For example, the *record_name*:

ES_NAME.1

with KEY=ES_NAME and CYCLE=1, will cause only one record to be transferred, while:

ES_NAME.1:10

with KEY=ES_NAME and CYCLES = 1 through 10 will cause records named ES_NAME.1, ES_NAME.2,..., ES_NAME.10 to be transferred. When using the *M2G directive, the *record_name* must be specified.

1.8.1.4.4 Macrosymbol Arithmetic

Macrosymbols may be used to perform arithmetic operations, producing new macrosymbols or explicit in-line numbers. Such macrosymbol operations are often used in command language procedures, either to control the runstream, or in preparation for numeric processor input. A summary of the basic binary arithmetic operations that may be performed on macrosymbols is given in Table 1.8-4.

Table 1.8-4 Arithmetic Operations With Macrosymbols

Operation	Description
< <*a*> + <*b*> >	Addition of two numeric macrosymbols, *a* and *b*
< <*a*> - <*b*> >	Subtraction of numeric macrosymbols, *b* from *a*
< <*a*> * <*b*> >	Multiplication of two numeric macrosymbols, *a* and *b*
< <*a*> / <*b*> >	Division of numbered macrosymbol *a* by *b*
< <*a*>% <*b*> >	Integer division of numeric macrosymbol *a* by *b*
< <*a*>^<*b*> >	Numeric macrosymbol *a* raised to the power *b*
<function(<*a*>,<*b*>,...)>	Evaluation of a macrosymbol function with macrosymbol arguments *a*, *b*, etc.

An example of using macrosymbol arithmetic to define a new macrosymbol would be:

*DEF/G *c* = < (<*a*> + (2.*<*b*>))^3 >

which defines a new floating point macrosymbol, *c*, to be equal to the sum of *a* plus twice *b*, all raised to the power 3, where *a* and *b* are previously-defined numeric macrosymbols. Outer angle brackets < > around arithmetic expressions are mandatory to force arithmetic evaluation. Inside the expression, parentheses may be used to indicate operational precedence, but angle brackets must be used to enclose macrosymbol names and force them to be decoded into numeric values.

1.8.1.4.5 Built-In Macrosymbols

COMET-AR has a number of built-in macrosymbols (and macrosymbol functions) which are described in detail in [2]. Two of the most commonly used built-in macrosymbols are TRUE and FALSE, which decode to 1 and 0, respectively (i.e., <TRUE>=1 and <FALSE>=0). A summary of the most commonly used built-in macrosymbols (constants and functions) is given in Table 1.8-5.

Table 1.8-5 Summary of Commonly Used Built-in Macrosymbols

Macrosymbol	Description
ABS(*a*)	Computes the absolute value of *a*
COS(*a*)	Computes the cosine of the angle *a* (radians)
COSD(*a*)	Computes the cosine of the angle *a* (degrees)
D2R	Conversion factor for degrees-to-radians = .01745329...
FALSE	Integer value associated with false logical expression = 0
IFELSE(*a*;*b*;*c*;*d*)	Compares *a* and *b*; if equal (or matching string) then it evaluates to *c*; else it evaluates to *d*
LOG(*a*)	Computes natural log (base e) of *a*

Table 1.8-5 Summary of Commonly Used Built-in Macrosymbols (Continued)

Macrosymbol	Description
MAX(*a*;*b*)	Computes maximum of *a* and *b*
MIN(*a*;*b*)	Computes minimum of *a* and *b*
MOD(*a*;*b*)	Computes modulus (remainder) of *a* divided by *b*
PI	Value of the constant $\pi = 3.14159...$
SIGN(*a*;*b*)	Computes absolute value of *a* times sign of *b*
SIN(*a*)	Computes the sine of the angle *a* (radians)
SIND(*a*)	Computes the sine of the angle *a* (degrees)
SQRT(*a*)	Computes square root of *a*
TAN(*a*)	Computes tangent of angle *a* (radians)
TAND(*a*)	Computes the tangent of the angle *a* (degrees)
TRUE	Integer value associated with true logical expression = 1

Other built-in macrosymbols may be found in Volume II (Directives) of reference [2].

1.8.1.5 Miscellaneous Directives

*1.8.1.5.1 The *ADD Directive*

The *ADD directive redirects command input to a file, much like a FORTRAN INCLUDE statement. It is also used to "compile" command language procedures before they can be called via the *CALL directive. The directive format is:

*ADD *filename*

where *filename* is the name of the file from which COMET-AR will begin reading input data. The added file may contain procedure definitions, calls to procedures defined in a file other than *filename*, and other *ADD directives. It may also contain processor input data (i.e., commands, data lines, etc.). For example, a user may want to use some pre-processor to generate files containing nodal locations and element connectivity. Once these files have been generated, they may be used as input for COMET-AR processors, such as TAB and ES*i*, by issuing the *ADD directive at the appropriate points in the runstream (i.e., model definition procedure).

When using the *ADD directive to compile a command-language procedure, the *filename* refers to the name of the file containing the procedure, and the output will be a compiled (i.e., callable) version of the procedure in a new file, whose name will be the procedure name appearing in the *PROCEDURE directive. Alternatively, if the *SET PLIB directive has been used, then the com-

piled (callable) procedure will be output to a record group in the indicated data library, with record name equal to the procedure name.

The input procedure file (associated with *filename*) can contain more than one command-language procedure. In this case, the output will be multiple compiled procedure files, or alternatively, multiple record groups on the data library indicated by the *SET PLIB directive.

The *ADD directive is thus useful for creating and updating selected procedures in a procedure data library (or procedure library). For example, you may make a copy of the standard COMET-AR procedure library ("proclib.gal"), which contains all of the procedures described in Part II of this manual, and then add additional procedures (e.g., for model definition), or update and replace existing procedures (e.g., solution procedures), by employing the *SET PLIB directive followed by the *ADD directive to compile and store/replace the new/modified procedures.

*1.8.1.5.2 The *ECHO,ON and *ECHO,OFF Directives*

These directives cause command and directive input to be either echoed or not echoed as it is being processed. The directive format is:

*ECHO,ON [,MA,MD] or *ECHO,OFF

where the optional MA and MD lead to detailed decoded printout of macrosymbol expressions.

*1.8.1.5.3 The *HELP Directive*

This directive provides on-line help on selected directives. The directive format is:

*HELP *directive_name*

where *directive_name* is the name of a valid directive (without the * prefix). The COMET-AR "Help File" must be properly installed before using the *HELP directive.

*1.8.1.5.4 The *SET and *SHOW Directives*

The *SET directive allows a number of intrinsic parameters to be changed from their default values. The *SET PLIB directive described under the Procedure Management directives, is one example of this generic directive. For a comprehensive list of other *SET directives, consult reference [2] (Volume II). Similarly, the *SHOW directive may be used to show the current settings. The following two *SHOW directives are extremely useful:

*SHOW MACROS [*macro_name(s)* /BI]

which prints the current values of macrosymbols indicated by *macro_name*(s), the /BI qualifier denotes built-in macrosymbols; and

```
*SHOW ARGUMENTS
```

which lists the procedure arguments in the current procedure, and prints their values (i.e., replacement text). Consult reference [2] for other options on the *SHOW directive.

*1.8.1.5.5 The *REMARK Directive*

The *REMARK directive is used to print out a remark while processing a COMET-AR procedure or ordinary command input file. The directive format is:

```
*REMARK remark
```

where *remark* may be any alphanumeric string and may contain embedded macrosymbol evaluations, including macrosymbol arithmetic. For example, the following is a valid remark:

*REMARK The result of multiplying <*a*> * <*b*> is < <*a*>*<*b*> >.

where *a* and *b* are previously-defined macrosymbols. If the values of *a* and *b* were 10 and 25, the above remark would lead to the following printed line at run-time.

The result of multiplying 10. * 25. is 250.

The *REMARK directive is useful for designing the output of a user-written command procedure. Such directives also appear in many standard COMET-AR solution procedures, indicating the current status of the solution as well as printing certain key parameters during the course of the run. By turning the command and directive echo off (with the *ECHO,OFF directive), the user will see only processor-based output and *REMARK-based output in the COMET-AR log file.

1.8.2 Processor Commands

Processor commands are input lines directed to specific COMET-AR processors, and are described in detail in Part II (Processors) of this manual. Processor commands typically begin with keywords (i.e., verbs), and may contain various qualifiers, keyword phrases and plain data, on one or multiple lines of input. While most processors have their own independent command language (in addition to the directives which are available while running any processor), there are a few common processor commands and conventions.

Since all COMET-AR processors employ the same command parser (CLIP), most of the basic syntactical conventions are uniform. The most important ones are described here.

1.8.2.1 Continuation Lines

Processor commands that require a single "logical" line of input may be continued on multiple "physical" lines by using a double-dash continuation mark (--). For example:

ELEMENT = 100 NODES = 1024, 1025, 2011, 2012, 2222, 3125, 4712 --

3025, 3022

would be interpreted as a single logical line by the receiving processor (in this case ES*i*).

1.8.2.2 Integer Sequence Format

Another common syntactical feature employed by COMET-AR processors is the "implied integer do-loop" convention, which expands expressions of the form:

i:j:k

to:

i, i+k, i+2k, i+3k, . . ., j

where i, j, and k represent integers; the default value of k is 1.

1.8.2.3 Separators: Commas, Spaces, and Semicolons

In general, commas and spaces are interchangeable as item separators. Refer to specific processor command descriptions. Semicolons must only be used in the following three situations:

1) to separate procedure arguments;
2) to separate arguments in macrosymbol functions; and
3) to separate multiple logical lines on the same physical line.

An example of case 3 would be:

FORM STIFFNESS ; FORM MASS ; FORM FORCE

which essentially enters three separate FORM commands on the same physical line.

1.8.3 Common Processor Commands

The following commands are common to all COMET-AR processors. All others are described in conjunction with specific processors in Part III of this manual.

1.8.3.1 The RUN Command

The RUN command is used to invoke a specific COMET-AR processor after running another COMET-AR processor. The command format is:

RUN *processor_name*

where *processor_name* is the name of the processor to be run. To use the RUN command, the user must initiate the COMET-AR execution with the COMET-AR macroprocessor described in Part III under the chapter on Special-Purpose Processors. Thereafter, the RUN command may be employed from within any COMET-AR processor, as it will cause control to first be transferred back to the COMET-AR macroprocessor before running the indicated processor. An exception to this rule is when the RUN command is issued to re-run the current processor without first issuing a STOP command, in which case control will remain with the current processor without intervention by the macroprocessor.

The RUN command may or may not actually cause execution of an independent processor, depending on how the COMET-AR macroprocessor is configured. Those COMET-AR processors that are embedded within the COMET-AR macroprocessor (a decision that can be made by the system administrator) will not be run as independent processors, but will simply be called as subroutines from within the COMET-AR macroprocessor. All other COMET-AR processors (which are external to the macroprocessor) are considered external processors, and will be executed independently by the macroprocessor upon issuance of the RUN command.

1.8.3.2 The STOP Command

The STOP command is used to properly terminate execution of the current COMET-AR processor. The command format is simply:

STOP

In general, use the STOP command for one processor before running another processor with the RUN command. It is especially important to issue a STOP command for the last processor in the current COMET-AR runstream.

1.8.3.3 The SET (or RESET) Command

Most COMET-AR processors have a command of the form:

SET *Parameter* =*Value(s)* **or** RESET *Parameter* =*Value(s)*

to set (or reset) various parameters to non-default values prior to issuing an action command.

1.9 Glossary of COMET-AR Terms, Notations, and Symbols

Tables 1.9-1 to 1.9-3 define COMET-AR terms, notations, and math symbols.

Table 1.9-1 Glossary of COMET-AR Terms

Term	Meaning
Adaptive Refinement	Refers to adaptive mesh refinement, wherein an initial finite element mesh is updated automatically to adapt to solution needs in a more-or-less optimal fashion, satisfying user accuracy requirements.
AR	Acronym for Adaptive Refinement.
AUTO_DOF_SUP	An analysis option that automatically suppresses extraneous DOFs that are not supported by element stiffness (e.g., drilling DOFs).
AUTO_DRILL	An analysis option that automatically adds artificial drilling stiffness only to those nodal DOFs that require it.
AUTO_TRIAD	An analysis option that automatically re-aligns the computational frames at nodes so that extraneous drilling DOFs can automatically be suppressed (via the AUTO_DOF_SUP option).
*call directive	A command-language directive used to call (i.e., invoke) another command-language procedure.
Case	An application problem. *Case* refers to the user-defined name for the application problem.
CLAMP	Acronym for Command Language for Applied Mechanics Processors; combination of procedure directives and processor commands that are parsed by the CLIP architectural utility in COMET-AR.
CLIP	Acronym for Command Language Interpretation Program; the architectural utility that parses COMET-AR's command language (see also CLAMP).
.clp files	COMET-AR command language procedure files; it is conventional to use .clp as a suffix for such files (unless they are embedded in a procedure library).
.com files	UNIX script files that are used to execute COMET-AR.
COMET-AR	Acronym for COmputational MEchanics Testbed.
COMET-AR User	Someone interested in performing an analysis with COMET-AR.
COMET-AR Developer	Someone participating in the extension of COMET-AR capabilities.
Command	In a command language: an instruction consisting of one or more items to be interpreted by the program that receives it.
Command Language	An interpretable language consisting of a stream of commands that controls the execution of a software system.
Computational Frame	Reference frame that defines DOF directions at each node.
Corotational Frame	Reference frame attached to each element; defines bulk rigid body motion, and facilitates treatment of large rotations in beams/shells.

Table 1.9-1 Glossary of COMET-AR Terms (Continued)

Term	Meaning
Database	One or more data files representing the definition of a COMET-AR model and/or solution.
Data Library	A term used to refer to a COMET-AR data file within a database.
Data Object	A tabular data structure that contains both data attributes, and utilities that perform operations on the data (see HDB).
DBC	Suffix used for main COMET-AR database file, as in Case.DBC.
DBE	Suffix used for COMET-AR file containing element matrices.
DBS	Suffix used for COMET-AR file containing system matrices.
DOF(s)	Degree(s) of freedom.
Drilling DOF	The DOF associated with rotation about the normal vector to the surface of a plate or shell element; in many shell element formulations, this DOF has no stiffness associated with it.
Drilling stiffness	The stiffness associated with the drilling DOF of a shell element. Many shell elements have no intrinsic stiffness associated with this DOF; some add artificial stiffness to stabilize it during the solution.
Element	Abbreviated term for finite element.
Error estimates	Typically refers to estimates of the discretization error in the solution for a given finite element mesh.
EltNam	Element name; the concatenation of the element processor name and the element type name, with an underscore (_) in between.
Generic Constitutive Processor (GCP)	A COMET-AR processor within which all constitutive models are implemented. The GCP appears both as a stand-alone processor (for material/fabrication definition) and as a utility library invoked by the Generic Element Processor (during the solution phase). May also be used for stand-alone analysis at a material point.
Generic Element Processor (GEP)	A software template (or "shell") for all COMET-AR structural element processors; provides a common generic user and developer interface to such processors. Also referred to as ES. Individual element processors have names that begin with ES (e.g., ES7p).
Global Frame	Fixed reference frame in which nodal coordinates are defined.
h refinement	Mesh refinement based on element subdivision.
hc refinement	Form of h refinement based on constraints to enforce inter-element compatibility between refined (i.e., subdivided) elements and unrefined elements.
hs refinement	Form of h refinement based on superposition of fine mesh regions on top of coarse mesh regions; a hierarchical version of hc ref.
ht refinement	Form of ht refinement based on the use of mesh transition patterns to connect refined element regions to unrefined element regions.
HDB	High-level database utility employed by COMET-AR to manage data objects within data files.
LDI (or ldi)	Acronym for logical device index.

Table 1.9-1 Glossary of COMET-AR Terms (Continued)

Term	Meaning
Local Frame	Reference frame attached to each element integration point; defines directions in which elt. strains are originally computed.
Logical Device Index	Positive integer used to identify data libraries currently attached to COMET-AR processors; used internally as a substitute for the data library's file name.
Macro-Processor	The COMET-AR processor that is used to start up the COMET-AR system, and from which other COMET-AR processors are executed (via the RUN command); may embed one or more other COMET-AR processors as internal processors for efficiency.
Procedure	A command language program written in COMET-AR's intrinsic language: CLAMP (sometimes referred to as CLIP, which is actually the utility that parses the CLAMP language).
Procedure Argument	A parameter specified in the header of a command-language procedure that may be used to replace text within the procedure.
Procedure Library	A special data library (i.e., file) that contains compiled COMET-AR command language procedures, ready to be invoked by users.
mesh	A given finite element discretization of an application problem.
*open	Command-language directive used to open old or new COMET-AR database files (i.e., data libraries), as in "*open ldi, dbname," where ldi denotes the logical device index and dbname denotes the file name.
RUN command	Special command recognized by COMET-AR to execute an individual COMET-AR processor, as in RUN processor_name.
Runstream	The collective set of UNIX script files and COMET-AR procedure files used as input to perform a particular analysis.
shell element	A structural element used to model thin or thick shell structures.
smoothing-based	Refers to error estimates that are based on comparing a discontinuous finite element stress field with a "smooth" version by nodal averaging.
Stress Frame	User-selected reference frames to be used for stress/strain output at element integration points, nodes, or centroids.
*stop directive	Command-language directive used to terminate COMET-AR, when executing the COMET-AR macro-processor.
User	See COMET-AR User.

Table 1.9-2 Glossary of COMET-AR Notation Conventions

Notation	Example	Meaning
Curly brackets	{ a, b, c, d }	Used to identify a list of related elements.
Square brackets (1)	[a, b, c, d]	When used in processor command syntax definitions, terms within square brackets are optional.
Square brackets (2)	RUN [proc_arg]	When appearing within command-language procedure, surrounding procedure argument names, square brackets indicate string replacement; i.e., the entire phrase [proc_arg] will be replaced by the value or string associated with the procedure argument "proc_arg" when the procedure was called (via the *call directive).
Vertical bars	GLOBAL { X | Y | Z }	When appearing in processor command syntax, or procedure argument syntax definitions, vertical bars indicate mutually exclusive options. In the example at left, only one of the terms X, Y or Z may be used with the GLOBAL phrase (e.g., GLOBAL X, GLOBAL Y, or GLOBAL Z).

Table 1.9-3 Glossary of COMET-AR Math Symbols

Symbol	Meaning
C	Constitutive matrix relating incremental strain to incremental stress.
D	Damping matrix for finite element model.
d	Displacement array for finite element model.
E	Total absolute error in strain energy norm of finite-element solution.
$\hat{E}$	Total relative error in strain energy norm; absolute error in finite element solution normalized by square root of total strain energy.
E_e	Element absolute error in energy norm (square root of strain energy).
$\hat{E}_e$	Element relative error in energy norm; element absolute error normalized by some measure of element strain energy norm.
f	Force vector for finite element model.
K	Stiffness matrix for finite element model.
M	Mass matrix for finite element model.
m	Shell (or beam) element bending-moment stress-resultants.
n	Shell (or beam) element force (membrane) stress-resultants.
N_a	Element shape function corresponding to element node "a."
Nel	Number of elements in the model.
q	Shell element transverse-shear-force stress-resultants.
Nen	Number of element nodes (per element).

Table 1.9-3 Glossary of COMET-AR Math Symbols (Continued)

Symbol	Meaning
U	Total strain energy.
U_e	Element strain energy for element "e."
U^{FE}	Total strain energy emanating from finite-element solution.
U^{SM}	Total strain energy emanating from smoothed finite-element solution.
$\hat{U}$	Strain energy density (strain energy per unit "volume").
$\hat{U}^{FE}$	Strain energy density emanating from finite element solution.
$\hat{U}^{SM}$	Strain energy density emanating from smoothed finite element solution.
ε	Element strain array.
ε^{FE}	Same as ε, but FE makes "Finite-Element" explicit.
ε^{SM}	Nodally smoothed version of ε, obtained by post-processing.
$\bar{\varepsilon}$	Shell (or beam) element reference-surface (membrane) strains.
κ	Shell (or beam) element change-of-curvature (bending) strains.
γ	Shell element transverse-shear strains.
σ	Element stress array.
σ^{FE}	Same as σ, but FE makes "Finite-Element" explicit.
σ^{SM}	Nodally smoothed version of σ, obtained by post-processing.
Ω	Problem domain represented by finite element model.
Ω_e	Element domain (for element e); may be volume, area, or line.

1.10 References

[1] Stehlin, B., et al., *The COMET-AR Tutorial*, preliminary NASA Contract Report, February 1993.

[2] Felippa, C., *The Computational Structural Mechanics (CSM) Testbed Architecture, Volume I: Language, Volume II: Directives, and Volume III: Fortran Interface*, NASA CRs 178383, 178384, and 178385, February 1989.

[3] Stanley, G. and Swenson, L., HDB: *High-Level (Object Oriented) Database Utilities for COMET-AR*, preliminary NASA CR, August 1992.

[4] Wright, M., Regelbrugge, M., and Felippa, C., *The Computational Structural Mechanics (CSM) Testbed Architecture, Volume IV: The Global Database Manager GAL-DBM*, NASA CR 178387, January 1989.

[5] Stehlin, B., *DB/MEM: Generic Database Utilities for COMET-AR*, preliminary NASA CR, May 1992.

[6] Stewart, C., *The Computational Structural Mechanics (CSM) Testbed User's Manual*, NASA TM 100644, October 1989.

[7] Stewart, C., *The Computational Structural Mechanics (CSM) Testbed Procedures Manual*, preliminary NASA TM, May 1990.

[8] Stanley, G. and Nour-Omid, *The Computational Structural Mechanics (CSM) Testbed Generic Structural-Element Processor Manual*, NASA CR 181728, May 1990.

[9] Hurlbut, *The Computational Structural Mechanics (CSM) Testbed Generic Constitutive Processor Manual*, NASA CR, May 1990.

Part II

PROCEDURES

In this part of the COMET-AR User's Manual, we describe available high-level comand-language procedures written in the CLAMP (Command Language for Applied Mechanics Processors) which may be invoked by the user. Some of these procedures, such as Model Definition procedures, can be written by the user. Others, such as Solution Procedures and Utility Procedures, are "canned" and so may be invoked directly by the user to perform various analysis functions.

Chapter 2 Model Definition Procedures

2.1 Overview

Model Definition Procedures are command-language (CLIP) procedures that generate all of the data associated with the initial mesh of a structural model. For reasonably simple models, Model Definition Procedures are typically written by the user. For more complicated models, these procedures may be generated automatically (or bypassed) by using PATRAN as a pre-processor, followed by the PATRAN-to-COMET-AR converter (PST) described in Part III. The main purpose of this chapter is to describe the ingredients of a typical Model Definition Procedure, so that the user may either construct a new one, or modify/interpret an existing one (some existing Model Definition Procedures are summarized in Section 2.11). This chapter may also be valuable to users employing PATRAN to generate the model, as certain COMET-AR modeling conventions must be understood in order to use the PATRAN-to-COMET-AR converter (see Section 2.12). The organization of this chapter is summarized in Table 2.1-1.

Table 2.1-1 Outline of Chapter 2: Model Definition Procedures

Section	Title
2.1	Overview
2.2	Reference Frames and Coordinate Systems
2.3	Generic Model Definition Procedures
2.4	Node Definition Procedures
2.5	Element Definition Procedures
2.6	Material/Fabrication Definition Procedures
2.7	Orientation of Fabrication Reference Frames
2.8	Load Definition Procedures
2.9	Boundary Condition Definition Procedures
2.10	Automatic DOF Suppression and Drilling Stabilization
2.11	Sample Model Definition Procedures (Summary)
2.12	Model Definition via PATRAN (and PST Translator)
2.13	Procedure GM2AM

Many of these sections refer to various COMET-AR processors described in Part III of this manual. Refer to the COMET-AR Tutorial for explicit examples of how to construct a Model Definition Procedure.

2.2 Reference Frames and Coordinate Systems

There are several reference frames and associated coordinate systems that the COMET-AR user should become familiar with before defining a model.f The most important of these are summarized in Table 2.2-1.

Table 2.2-1 COMET-AR Reference Frames

Reference Frame	Coordinate Axes	Role in COMET-AR
Global	x_g, y_g, z_g	Fixed frame used for defining initial nodal coordinates; also the default frame for orienting nodal DOFs.
Computational (Nodal)	x_c, y_c, z_c	Used for orienting nodal DOFs; may vary from node to node, or may be fixed. Selected during the Node Definition phase. (Default: Global Frame).
Corotational (Element)	x_e, y_e, z_e	Used internally by generic element processor to track element "rigid body" motion, and subtract it from total deformations before computing strains. Varies from element to element, but constant within a given element.
Local (Integ. Point)	x_l, y_l, z_l	Used by elements to express strains and stresses at element integration points unless an alternate Stress Frame is selected by the user. May vary from integration point to integration point.
Stress (Integ. Point)	x_s, y_s, z_s	Rotated version of the Local Frame used for database stress and strain output; selected via STR_DIR argument in solution procedures.
Fabrication (Integ. Point)	x_f, y_f, z_f	Used to orient material fabrications, such as laminated composite layups, in space. May vary from integration point to integration point. Selected via the FAB_DIR subcommand of the DEFINE ELEMENTS command in the generic element processor.
Material (Integ. Point)	x_m, y_m, z_m	Used to orients individual material fibers comprising a fabrication, e.g., each layer in a composite laminate is oriented via a fiber angle, θ_{mf}, between the Fabrication Frame and the Material Frame of that layer.

Each of the reference frames in Table 2.2-1 is orthogonal (i.e., the corresponding x, y, and z axes are mutually perpendicular and form a right-handed system, or triad). An illustration of these various reference frames and how they relate to one another in a simple model is given in Figure 2.2-1.

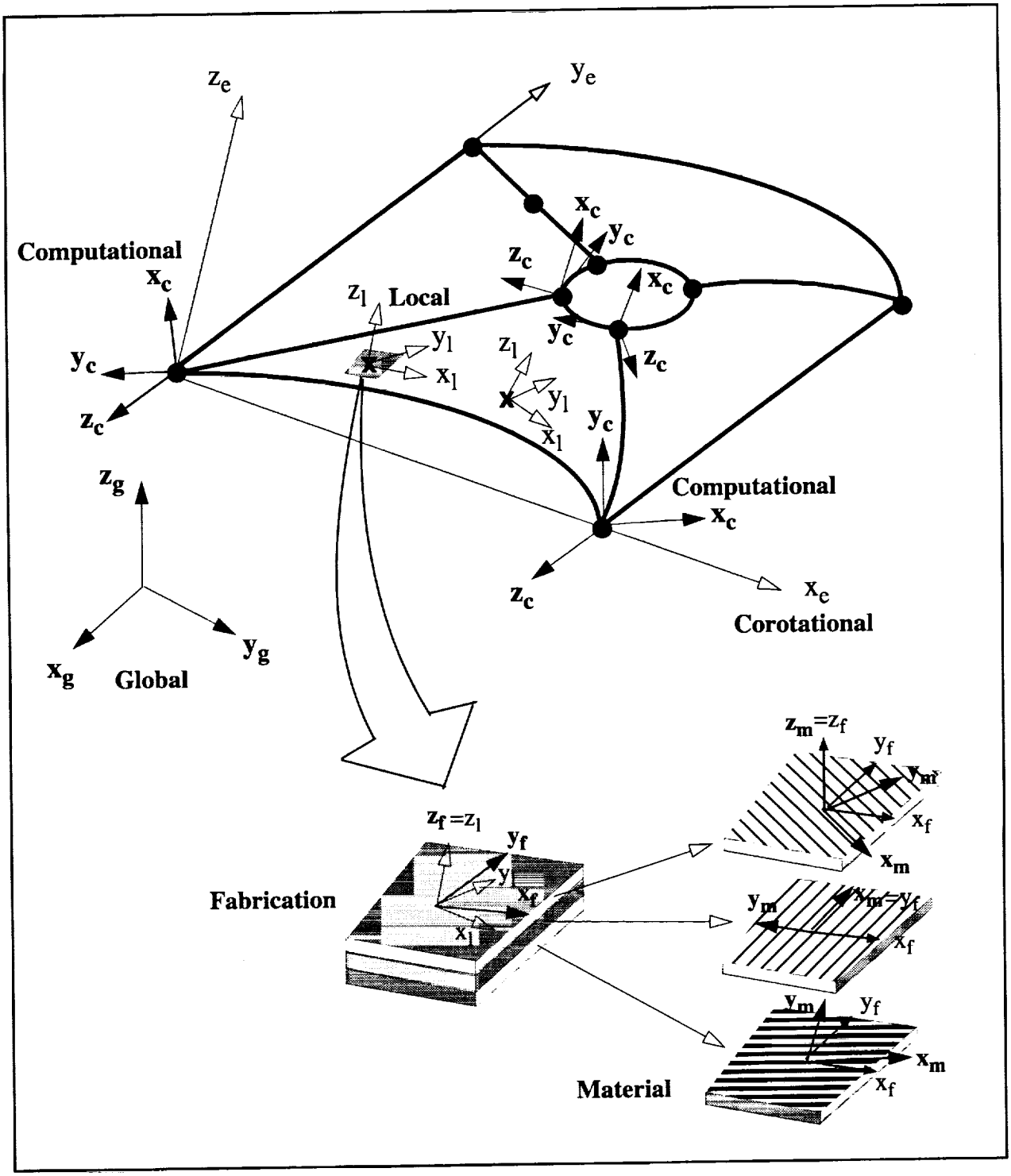


Figure 2.2-1 Example of COMET-AR Reference Frames

2.2.1 Global Frame

The Global Frame is represented by a fixed Cartesian coordinate system, x_g, y_g, z_g, that is the default system used to define nodal coordinates and to orient nodal (or computational) reference frame DOFs in the initial configuration of the structural model. The user may also employ a global cylindrical coordinate system to define nodal coordinates and computational frames in TAB (see Section 6.10). Alternate nodal DOF reference frames may be defined in TAB as well.

2.2.2 Computational Frame

The Computational Frame is represented by a nodally varying orthogonal triad, x_c, y_c, z_c, and is used to express the components of the nodal (displacement) DOFs, including both translations and rotations. These nodal frames are typically defined by the user (in Processor TAB) to facilitate the definition of boundary conditions and/or the interpretation of displacement results. The default Computational Frame at a node is the Global Frame. Alternate Computational Frames may be defined via the ALTREF command in Processor TAB, or if a cylindrical coordinate system has been selected for node definition, that cylindrical system may be used to automatically generate a local, cylindrically-aligned triad as the Computational Frame at each node.

> The Computational Frame at a given node is defined in the initial configuration and fixed throughout the motion of the structural model (i.e., it does not rotate to follow the deformed configuration). This must be taken into account when specifying nodal boundary conditions.

2.2.3 Corotational Frame

The Corotational Frame is represented by a separate orthogonal triad, x_e, y_e, z_e, that is attached to each element in the model. This triad is defined automatically (by the Generic Element Processor, ES) in the initial element configuration, and rotates with the "rigid body" part of the element deformation. During element strain computation, this rigid body motion is subtracted from the nodal displacements leaving "deformational displacements" whose rotatational components are much smaller than the total nodal rotations. This enables arbitrarily large total rotations to be handled by shell elements that are based on only moderate (or even infinitesimal) rotation theories. The Corotational Frame is also used to form element stiffness and force arrays within an element processor, although all element stiffness and force arrays are output to the database in computational components, x_c, y_c, z_c.

For shell (2D) and solid (3D) elements, the Corotational Frame is defined as an orthogonal triad aligned with an average plane passing through the first three or four element corner points for triangular and quadrilateral planform elements, respectively (see Figure 2.2-1). For beam (1D) elements, the Corotational Frame is initially oriented via the beam element reference node.

2.2.4 Local Frame

The Local Frame is represented by an orthogonal triad, x_l, y_l, z_l, that is attached to each element integration point in the initial configuration. This triad represents the intrinsic directions (at an element integration point) used to express strain and stress components within an element processor. The orientation of the Local Triad may vary from integration point to integration point within an element (see Figure 2.2-1) or may be the same at all integration points for some elements (e.g., triangular shell elements). The definition of x_l, y_l, z_l is dependent on element type and described within specific element processors (see Chapter 7, *Element Processors*). Before strains and stresses are output to the database, they may be transformed to an alternate "Stress Frame."

2.2.5 Stress Frame

The Stress Frame is a rotated version of the Local Frame used exclusively to express strain and stress components stored in the database (for post-processing). The user may select the Stress Frame via the STR_DIR argument provided by Solution Procedures (see also the RESET STR_DIR command in the section on Processor ES). Current options include using the element Local Frame (the default option) or using the Fabrication Frame.

2.2.6 Fabrication Frame

The Fabrication Frame is represented by an orthogonal triad, x_f, y_f, z_f, that orients the fabrication definition (i.e., cross-section/materials) at each element integration point. For example, a layered-shell fabrication is illustrated in Figure 2.2-1 which has a Fabrication Frame with the x_f, y_f axes in the lamina plane, and the z_f axis normal to the laminate (i.e., the element reference surface). For beam fabrications, the z_f axis is parallel to the beam axis; and for solid fabrications, the x_f, y_f, z_f axes coincide with the material axes. Fabrications and associated materials are defined via the Generic Constitutive Processor (see Chapter 8, *Constitutive Processors*) and that the orientation and eccentricity of the fabrication are defined as element properties via the FAB_DIR and FAB_ECC subcommands under the DEFINE ELEMENTS command in the Generic Element Processor (see Chapter 7, *Element Processors*).

2.2.7 Material Frame

The Material Frame is represented by an orthogonal triad, x_m, y_m, z_m, that orients the material properties within a given fabrication. For layered-shell fabrications, the Material x_m,y_m axes are in a plane that is parallel to the Fabrication x_f,y_f axes, but rotated by a layer angle, θ_{mf}. For beam and solid fabrications, the Material and Fabrication frames are parallel.

> For shell elements, the surface-normal directions of the Local, Stress, Fabrication, and Material Frames all coincide (i.e., z_l=z_s=z_f=z_m).

2.3 Generic Model Definition Procedures

A Model Definition Procedure is a command-language (CLIP) procedure written by the user (or a pre-processing converter such as Processor PST) to generate a model in the COMET-AR database. This section describes the generic format of Model Definition Procedures, with more details given in subsequent sections. This format is not rigid, but it does establish the required ingredients and provide guidelines and a template for the user. For more detailed examples, refer to Section 2.11, *Sample Model Definition Procedures* or to the COMET-AR Tutorial. A generic Model Definition Procedure is shown in Box 2.3-1.

Box 2.3-1 Generic Model Definition Procedure

PROCEDURE** ***Case *(model_arg_1=model_def_1; . . . model_arg_n=model_def_n)*

. Open New Database

*OPEN *Case*.DBC

. Node Definition

. Nodal Coordinates and Computational Frames are defined here with Processor TAB.
. Typically done via a sub-procedure; e.g., ***CALL NodeDefn (. . .)**

. Element Definition

. Element Types, Nodal Connectivity, Fabrication Numbers, and Fabrication Frames/Eccentricity are defined here with Element (ES*i*) Processors.
. Typically done via a sub-procedure; e.g., ***CALL EltDefn (. . .)**

. Material/Fabrication Definition

. Fabrication and Material Properties are defined with the Generic Constitutive Processor (GCP).
. Typically done via a sub-procedure; e.g., ***CALL MatlDefn (. . .)**

. Load Definition

. Nodal (Concentrated) Loads and Element (Distributed) Loads are defined here with Processors AUS and ES*i*, respectively;
. Typically done via a sub-procedure; e.g., ***CALL LoadDefn (. . .)**

. Boundary Condition Definition

. Nodal Boundary Conditions are defined here via Processor COP.
. Typically done via a sub-procedure; e.g., ***CALL BCsDefn (. . .)**

***END**

In Box 2.3-1, *Case* represents the case name, a user-selected name for the model being defined. The phrase *model_arg_i* represents the *i*th user-defined procedure argument name, and *model_def_i* represents the corresponding default value (a number or character string). Such procedure arguments allow the use of arbitrary parameters that provide a convenient parametrization of the model. For example, the procedure may include such arguments as model dimensions, material properties (or numbers), element type, and initial mesh density.

The first statement in a Model Definition Procedure is typically an *OPEN directive which creates a new computational database file. The database file name should start with the *Case* name and end with the suffix .DBC. This ensures compatibility with Solution Procedures such as AR_CONTROL.

> While it is the responsibility of the user to create the initial *Case*.DBC file, it is not mandatory to include the *OPEN directive in the model definition procedure. Instead, the *OPEN can be placed within the Model.com (UNIX script) file that invokes the model definition procedure (see Section 1.5, *Execution of COMET-AR*).

As shown in Box 2.3-1, after creating an initial database file (*Case*.DBC), the Model Definition Procedure is typically composed of the following five functions: 1) Node Definition; 2) Element Definition; 3) Material/Fabrication Definition; 4) Load Definition; and 5) Boundary Condition Definition. For convenience (and readability) each of these model definition functions may be treated as a subprocedure (i.e., a lower-level procedure called from within the main model definition procedure) as described in the following subsections.

Once the entire Model Definition Procedure (including all internal sub-procedures) has been written by the user (or the PATRAN converter), it must be compiled and invoked from within a "Model.com" file as explained in Section 1.5.3, *Execution of COMET-AR/Pre-Processing Phase*.

2.4 Node Definition Procedures

The Node Definition part of a Model Definition Procedure is described in this section. Node Definition includes the definition of nodal coordinates and computational frames, and is performed via Processor TAB, and followed by Processor REDO (to reformat certain datasets). The necessary Processor commands can either be added directly (in-line) to the Model Definition Procedure or placed in a sub-procedure. The example in Box 2.4-1 employs a sub-procedure.

In the procedure shown in Box 2.4-1, (arbitrarily called NodeDefn), a number of Model Definition Procedure argument values have been transferred from above (i.e., *model_arg_j1 ... model_arg_jn*). These values are now referred to with the local arguments *node_arg_1 ... node_arg_n* and may be employed within the Node Definition Procedure by using square brackets for symbolic replacement (e.g., [*node_arg_1*]).

Box 2.4-1 Sample Node Definition Sub-Procedure

***PROCEDURE NodeDefn** (*node_arg_1* [=*model_val_j1*]; ... *node_arg_n* [=*model_val_jn*])

. **Basic Node Definition**

RUN TAB

. **Nodal Summary**

START *nn*

. **Nodal Coordinates**

JLOC

1 x_1 y_1 z_1

2 x_2 y_2 z_2

:

nn x_{nn} y_{nn} z_{nn}

. **Nodal Computational Frames**

. See ALTREF and JREF commands under Processor TAB

STOP

. **Reformat Model Summary and Nodal Datasets**

RUN REDO

CSM ; NCT ; NTT

STOP

***END**

The TAB Processor appearing in the above procedure is used to define both nodal coordinates (via the JLOC command), and nodal computational reference frames (via the ALTREF and JREF com-

mands). The TAB START command must first be used to specify the total number of nodes in the initial model (*nn*). The JLOC ("joint location") command is followed by a nodal coordinate line for each node, which includes the node number followed by the coordinate values. In the example shown in Box 2.3-1, global Cartesian coordinates are employed; however, there are other options available in TAB (e.g., cylindrical).

After nodal coordinates and computational reference frames are defined with Processor TAB, the user must employ Processor REDO to reformat certain datasets from the old (COMET-BL) data structures to the new (COMET-AR) data structures. The standard commands required to do this are shown in Box 2.3-1.

> Refer to Section 6.10 on Processor TAB for details on both nodal coordinate and computational frame definition, including the START, JLOC, JREF, and ALTREF commands shown in Box 2.4-1. Refer to Section 6.7 for details regarding the REDO commands shown in Box 2.3-1.

2.5 Element Definition Procedures

2.5.1 General Description

The Element Definition part of a Model Definition Procedure is described in this section. Element Definition includes the definition of element node, fabrication, and (optionally) solid-model geometry connectivity for all element types to be present in the model. These functions are all performed by structural element processors (ES*i*) that share a common user- and database-interface known as the Generic Element Processor (or ES) described in Section 7.2. The commands for element definition can either be added directly to the Model Definition Procedure or placed in a sub-procedure. The example in Box 2.5-1 employs a sub-procedure.

Box 2.5-1 Sample Element Definition Sub-Procedure

```
*PROCEDURE EltDefn ( elt_arg_1 [ = model_val_j1 ]; ... elt_arg_n [ = model_val_jn ] )
    RUN ESi1    . Define Elements of First Type ( Name = ESi1_EltTyp1 )
                RESET ELEMENT_TYPE = EltTyp1
                DEFINE ELEMENTS [ /P=p ]
                    FAB_ID = 1 ; FAB_DIR = GLOBAL X
                    ELEMENT = 1   NODES = n1^1, n2^1, . . ., nnen^1
                    ELEMENT = 2   NODES = n1^2, n2^2, . . ., nnen^2
                    :
                    ELEMENT = nel1   NODES = n1^nel1, n2^nel1, . . ., nnen^nel1
                END DEFINE ELEMENTS
    RUN ESi2    . Define Elements of Second Type ( Name = ESi2_EltTyp2 )
                RESET ELEMENT_TYPE = EltTyp2
                DEFINE ELEMENTS [ /P=p ]
                    ELEMENT = 1   NODES = n1^1, n2^1, . . ., nnen^1
                    ELEMENT = 2   NODES = n1^2, n2^2, . . ., nnen^2
                    :
                    ELEMENT = nel2   NODES = n1^nel2, n2^nel2, . . ., nnen^nel2
                END DEFINE ELEMENTS
    : : :
    STOP
*END
```

In the procedure shown in Box 2.5-1, (arbitrarily called *EltDefn*), a number of Model Definition Procedure argument values have been transferred from above (i.e., *model_arg_j1 ... model_arg_jn*). These values are now referred to with the local arguments *elt_arg_1 ... elt_arg_n* and may be employed within the Element Definition Procedure by using square brackets for symbolic replacement (e.g., [*elt_arg_1*]).

There must be a separate element processor (ES*i*) RUN statement for each element type appearing in the model. In this example, ES*i*1 denotes the element processor containing the first element type *EltTyp1*. ES*i*2 denotes the element processor containing the second element type, *EltTyp2*.

The combination of the element processor name (ES*i*) and the element type name within that processor (*EltTyp*) is called the "Element Name" (or *EltNam*), i.e.,

EltNam = ES*i*_*EltTyp*

The element name (*EltNam*) appears as a prefix in all element datasets. For example, the element definition dataset is called:

EltNam.DEFINITION...*mesh*

where *mesh* is the current mesh number. This combined element name provides a unique labeling of element types within COMET-AR and allows different element processors to have elements with the same element type name (*EltTyp*) since element processor names are always unique. Individual element processors (ES*i*) may contain multiple element types; thus, the combined name (*EltNam*) is both necessary and sufficient for unambiguous selection of an element type by the user.

After each RUN ES*i* statement, a RESET ELEMENT_TYPE command must be used to select the element type (*EltTyp*) within element processor ES*i* (as element processors may have multiple element types). Then the DEFINE ELEMENTS command is used to initiate the definition of element node and fabrication connectivity for elements of the specified type in the model (the optional /P=p qualifier must be appended to the command if the element processor contains a specific element type that permits variable polynomial orders, e.g., Processor ES7p). Subcommands under the DEFINE ELEMENTS apply to all elements that appear in subsequent ELEMENT subcommands and include: FAB_ID, which selects a fabrication (i.e., cross-section/material) type number; and FAB_DIR, which indicates how the fabrication is to be oriented. (Other optional subcommands that do not appear in the above example include the specification of element GROUP numbers and solid-model SURFACE connectivity).

The ELEMENT subcommand is then used to define node (and optionally solid-model line) connectivity, via the NODE phrase (and optional LINE phrase). The elements may be defined in an arbitrary order, as their position in the database is determined by the ELEMENT=*eltnum* phrase. For details on the DEFINE ELEMENTS command (and its subcommands) refer to the section on Processor ES. For details on individual element processors and element types, refer to the appropriate sections on Processor ES*i* (where *i* denotes the variable part of the name).

After defining all elements for all element types relevant to the current model, a STOP command should appear before the *END directive in the Element Definition Procedure.

2.5.2 Available Element Processors and Types in COMET-AR

A summary of element processors and element types currently available in COMET-AR is given in the Table 2.5-1.

Table 2.5-1 Summary of COMET-AR Element Processors/Types

Element Processor	Element Type	Description
ES1	EX41-46 EX91-96 EX47 EX97	Assorted 4-node selectively-reduced integrated shell elements. Assorted 9-node selectively reduced integrated shell elements. Basic 4-node ANS shell element. Basic 9-node ANS shell element.
ES1p	SHELL	Variable-order polynomial, assumed-displacement Lagrange (LAG) isoparametric quadrilateral shell elements: p=1: 4-node bilinear geometry and displacements p=2: 9-node biquadratic geometry and displacements p=3: 16-node bicubic geometry and displacements
ES5	E410	STAGS 4-node Kirchhoff-type shell element.
ES6	E210	STAGS 2-node Euler beam element.
ES7p	SHELL	Variable-order polynomial Assumed Natural-coordinate Strain (ANS) quadrilateral shell elements: p=1: 4-node bilinear geometry, const./linear strains p=2: 9-node biquadratic geometry, linear/quadratic strains p=3: 16-node bicubic geometry, quadratic/cubic strains
ES36	MIN3	Anisoparametric 3-node triangular shell element.
	MIN6	Extension of MIN3 to curved geometry (under development).

Detailed descriptions (and usage guidelines) for each of the above elements may be found within the corresponding sections in Chapter 7. Additional solid element processors are implemented but untested.

2.6 Material/Fabrication Definition Procedures

2.6.1 General Description

The Material/Fabrication Definition part of a Model Definition Procedure is described in this section. Material/Fabrication Definition includes the definition of fabrication properties (cross-section geometries and associated material numbers and orientations associated primarily with beam and shell elements); and material properties (material constants associated with specific constitutive models). The definition of both sets of properties is performed via the Generic Constitutive Processor (GCP) described in Chapter 8. The commands for material/fabrication definition can either be added directly to the Model Definition Procedure, or placed in a sub-procedure. The example in Box 2.6-1 employs a sub-procedure.

Box 2.6-1 Sample Material/Fabrication Definition Sub-Procedure

```
*PROCEDURE MatlDefn ( matl_arg_1 [ = model_val_j1 ]; ... matl_arg_n [ = model_val_jn ] )

  . Run Generic Constitutive Processor (GCP) and Define all Fabrication & Material Props.
      RUN GCP
          FABRICATION
          . Definition of one or more fabrications (FABID=1, 2, ...) goes here.
              :
          ENDFAB

          MATERIAL
          . Definition of one or more materials (MATID=1, 2, ...) goes here.
              :
          ENDMAT
      STOP

*END
```

In the procedure shown in Box 2.6-1 (arbitrarily called *MatlDefn*), a number of Model Definition Procedure argument values have been transferred from above (i.e., *model_arg_j1 ... model_arg_jn*). These values are now referred to with the local arguments *matl_arg_1 ... matl_arg_n* and may be employed within the Material/Fabrication Definition Procedure by using square brackets for symbolic replacement (e.g., [*matl_arg_1*]). Typically, these procedure arguments will be used to pass user-selected material properties, fabrication properties, or just material/

fabrication numbers enabling the user to select from a variety of property sets pre-defined within procedure *MatlDefn*.

There are two relevant top-level commands within the GCP for fabrication and material property definition: FABRICATION and MATERIAL. The FABRICATION command is used to initiate the definition of one or more sets of fabrication properties. Each set of fabrication properties has an associated fabrication type (e.g., SHELL) and fabrication number (e.g., FABID=1, 2, ...). The available GCP subcommands to define properties for specific fabrication types are described in Section 8.3. All fabrication types have one thing in common: they refer to one or more material numbers (MATIDs), the properties for which are defined via the MATERIAL command.

The MATERIAL command is used to initiate the definition of one or more sets of material properties. Each set of material properties is associated with a specific material type (e.g., ISOEL: isotropic elastic) and material number (e.g., MATID=1, 2, ...). The available GCP subcommands to define constitutive properties for specific material types are described in Section 8.4. The GCP supports either direct input of material property data, or the tabulation of predefined material properties in a material database.

> In the Element Definition Procedure (see Section 2.5) elements refer to fabrication numbers (FABIDs) and not directly to material numbers (MATIDs) when they are defined (via the FAB_ID subcommand of the DEFINE ELEMENTS command within the Generic Element Processor). In turn, fabrications refer to material numbers within the FABRICATION command of the Generic Constitutive Processor. The hierarchy is: materials belong to fabrications which in turn belong to elements.

> The separation of the Generic Constitutive Processor from element processors (ES*i*) as an independent module is a unique feature of COMET-AR. It makes existing and new material/fabrication types accessible to all element types simultaneously, avoids duplication of effort (and errors) by element developers, and allows constitutive developers and element developers to focus independently on their areas of expertise.

2.6.2 Available Fabrication Types in COMET-AR

A summary of fabrication types currently available in COMET-AR is given in Table 2.6-1.

Table 2.6-1 Summary of COMET-AR Fabrication Types (within the GCP)

Fabrication Type	Description
BEAM	Homogeneous beam element cross-section properties; includes geometric properties (area, moments of inertia, eccentricities) and an associated material number (MATID).
SHELL	Layered (composite) shell through-thickness properties; includes geometric properties (number of layers, layer thicknesses, layer fiber angles) and associated material numbers (MATIDs) for each layer.
SOLID	Three-dimensional solid continuum; includes only a material number (MATID), no geometric properties.

Detailed descriptions for each of the above fabrication types may be found in Section 8.3.

2.6.3 Available Material Types in COMET-AR

A summary of material types currently available in COMET-AR is given in Table 2.6-2.

Table 2.6-2 Summary of COMET-AR Material Types (within the GCP)

Material Type	Description
ISOEL	Isotropic elastic material; includes standard material constants (and optional temperature and moisture dependent parameters).
ORTEL	Orthotropic elastic material; includes standard material constants.
PLASTIC_WB	White-Besseling (mechanical sublayer) elastic-plastic constitutive model for initially isotropic materials (temperature-independent).

Detailed descriptions for each of the above material types may be found in Section 8.4.

2.7 Orientation of Fabrication Reference Frames

2.7.1 General Description

Fabrication (and embedded material) reference frames (x_f, y_f, z_f) are defined during element definition via the FAB_DIR subcommand of the generic element processor's DEFINE ELEMENTS command (see Section 2.5). The various options for orienting the fabrication frame are summarized in Table 2.7-1.

Table 2.7-1 Fabrication Frame Orientation Options

Option	Input_Data	Interpretation
ELEMENT	None	The fabrication frame is parallel to the element local frame. $x_f = x_l$, $y_f = y_l$, $z_f = z_l$. This option is useful only for very simple models with rectangular meshes.
GLOBAL	{ X | Y | Z }	The fabrication frame is such that the x_f axis is parallel to the global X (i.e., x_g), Y (i.e., y_g) or Z (i.e., z_g) axis. The z_f axis is taken parallel to the element normal (z_f) axis (for shells). The y_f axis follows from the right-hand rule. This option is useful for simple cylindrical structures where one of the global axes aligns with a structural direction of interest.
POINT	$\bar{x}$ [, θ]	The element local z_l axis is used for z_f. The y_f axis is obtained by taking the cross-product of the vector connecting the reference point $\bar{x}$ to the current element integration point, with z_l. The y_f axis follows from the right-hand rule. The angle θ is an arbitrary in-plane rotation (about z_f) that may be performed after the triad has been projected to the element tangent plane (x_l - y_l). This option is useful for axisymmetric shell structures, especially where annular plates are involved.
VECTOR	$\bar{v}$ [, θ]	The element local z_l axis is used for z_f. The y_f axis is obtained by crossing the reference vector $\bar{v}$ with z_m and x_m follows from the right-hand rule. This option is extremely powerful for general shell structures, where a different reference vector may be defined for each substructure, typically along a generator. The arbitrary in-plane angle, θ, may be used, e.g., to define a helical laminated composite winding on a cylindrical shell.
PLANE	$\bar{u}$, $\bar{v}$ [, θ]	First, a preliminary x_f', y_f', z_f' triad is constructed by taking x_f' parallel to $\bar{u}$, crossing $\bar{u}$ with $\bar{v}$ to obtain z_f', and crossing z_f' with x_f to obtain y_f. For shells, this triad is then projected onto the element tangent plane by rotating z_f' into the element normal axis z_l to obtain x_f, y_f, z_f. Finally, an optional in-plane rotation θ is provided. The PLANE option is useful for general 3D models.
BEAM	*node*	Node number of beam element reference point.

The above options are selected via the element processor FAB_DIR subcommand, i.e.,

FAB_DIR = *Option, Input_Data*

where *Option* is the option name, and *Input_Data* are the associated parameters.

For shell elements, the options listed in Table 2.7-1 employ the local element normal vector (parallel to z_l) to construct a tangent plane. The user may specify an additional arbitrary "in-plane" angle to rotate the fabrication frame after it has been aligned with the shell element local tangent plane (which can be useful for fiberwound composites on curved surfaces). The above options are not limited to shell elements. They may also be used with 3D solid elements, in which case the use of the element local normal (z_l) axis is optional.

These fabrication reference frame options may be used to vary the orientation from element to element, or among groups of elements, as indicated in Figure 2.7-1.

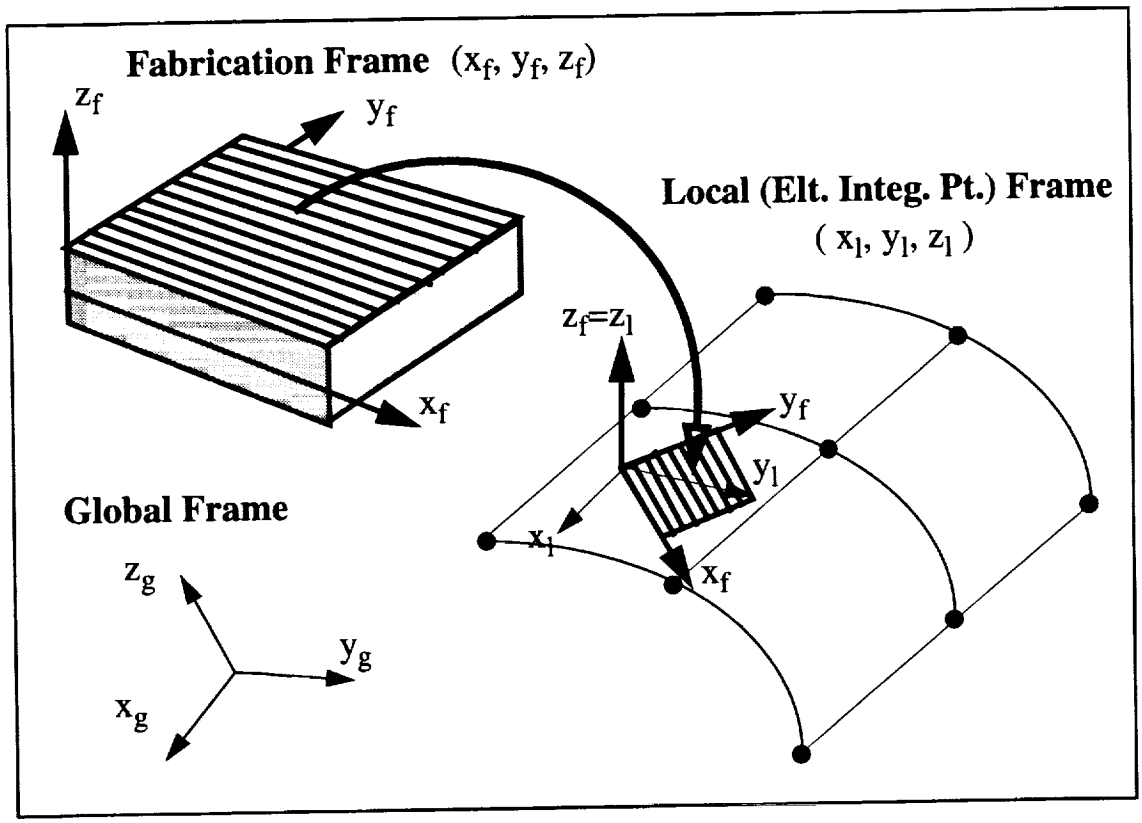


Figure 2.7-1 Orientation of Fabrication and Related Reference Frames

Finally, the fabrication reference frame may also be used for stress output, i.e., the element stress reference frame can be equated to the fabrication reference frame by setting the stress direction solution procedure argument to FAB_DIR:

```
STR_DIR = FAB_DIR
```

The following subsections describe the fabrication frame orientation options in more detail.

2.7.2 The FAB_DIR = ELEMENT Option

The ELEMENT option for defining fabrication reference frames is the default option and is invoked by issuing the DEFINE ELEMENTS subcommand.

```
FAB_DIR = ELEMENT
```

with no additional parameters required. The result is that the fabrication frame is equivalenced to the local element integration point frame, i.e., $x_f = x_l$, $y_f = y_l$, and $z_f = z_l$.

2.7.3 The FAB_DIR = GLOBAL Option

The GLOBAL option is invoked by issuing the DEFINE ELEMENTS subcommand.

```
FAB_DIR = GLOBAL { X | Y | Z } [ /3D ]
```

where specification of X, Y, or Z indicates that the fabrication x_f axis is parallel to the global x_g, y_g or z_g axis. For shell elements, the fabrication z_f axis is automatically parallel to the shell element normal vector (i.e., to the z_l axis), and the y_f axis completes a right-handed orthogonal triad. For solid elements (with the /3D qualifier), the y_f and z_f axes are defined by cyclic permutation of the global axis selected for x_f. The GLOBAL definition option is illustrated geometrically and mathematically in Figure 2.7-2.

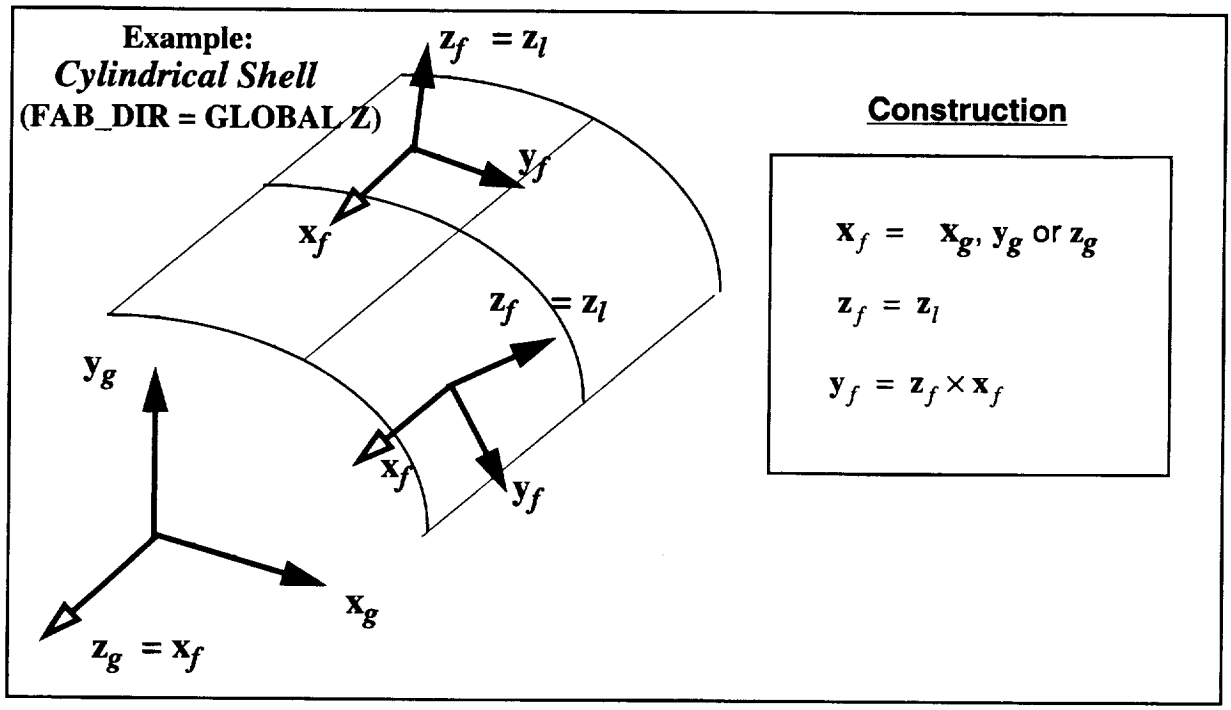


Figure 2.7-2 FAB_DIR = GLOBAL Option

2.7.4 The FAB_DIR = POINT Option

The POINT option is invoked by issuing the DEFINE ELEMENTS subcommand.

FAB_DIR = POINT $\bar{\mathbf{x}}$ [, θ]

where $\bar{\mathbf{x}}$ denotes the global coordinates of an arbitrary reference point, i.e., $\bar{\mathbf{x}} = \bar{x}_g, \bar{y}_g, \bar{z}_g$, from which a vector is connected to the current element integration point. This vector is then crossed with the element normal vector ($z_l = z_f$) to obtain the y_f direction, and the x_f direction is obtained via the right-hand rule. An optional in-plane rotation of θ is then performed about the z_f direction to obtain the final orientation of the fabrication frame. The construction is illustrated in Figure 2.7-3. The POINT option is particularly useful for annular and circular plate structures.

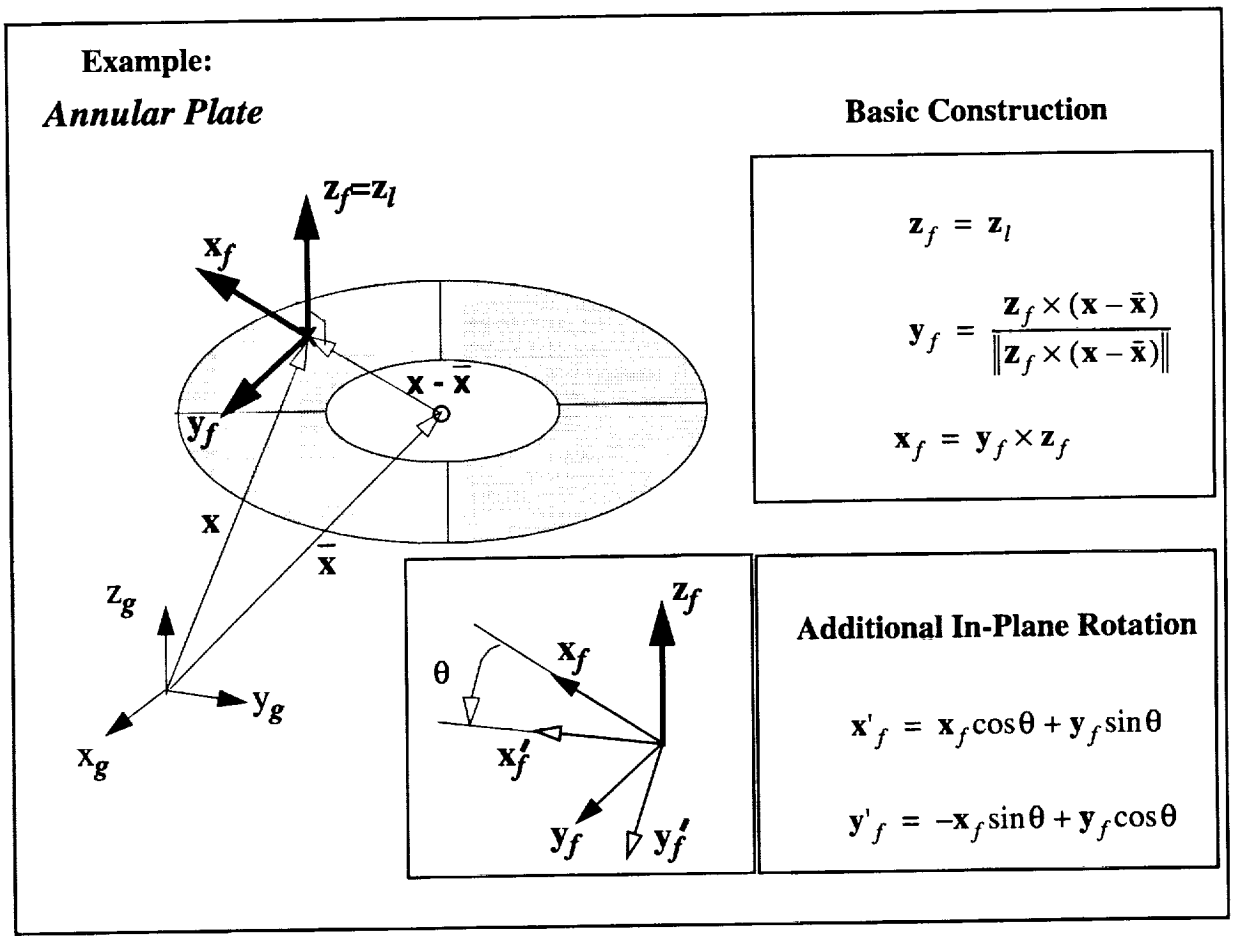


Figure 2.7-3 FAB_DIR = POINT Option

2.7.5 The FAB_DIR = VECTOR Option

The VECTOR option is invoked by issuing the DEFINE ELEMENTS subcommand

FAB_DIR = VECTOR $\bar{v}$ [, θ]

where $\bar{v}$ denotes the global components of a an arbitrary reference vector, i.e., $\bar{v} = \bar{v}_{xg}, \bar{v}_{yg}, \bar{v}_{zg}$, which is crossed with the element normal vector ($z_l = z_f$) to obtain the y_f direction; the x_f direction is obtained via the right-hand rule. An optional in-plane rotation of θ is then performed about the z_f direction to obtain the final orientation of the fabrication frame. This option may be used to generate fabrication frame triads for general shell structures. It is particularly useful for assemblages of shells of revolution, where the axes of revolution (i.e., the shell generators) provide natural reference vectors, $\bar{v}$. The construction is illustrated in Figure 2.7-4.

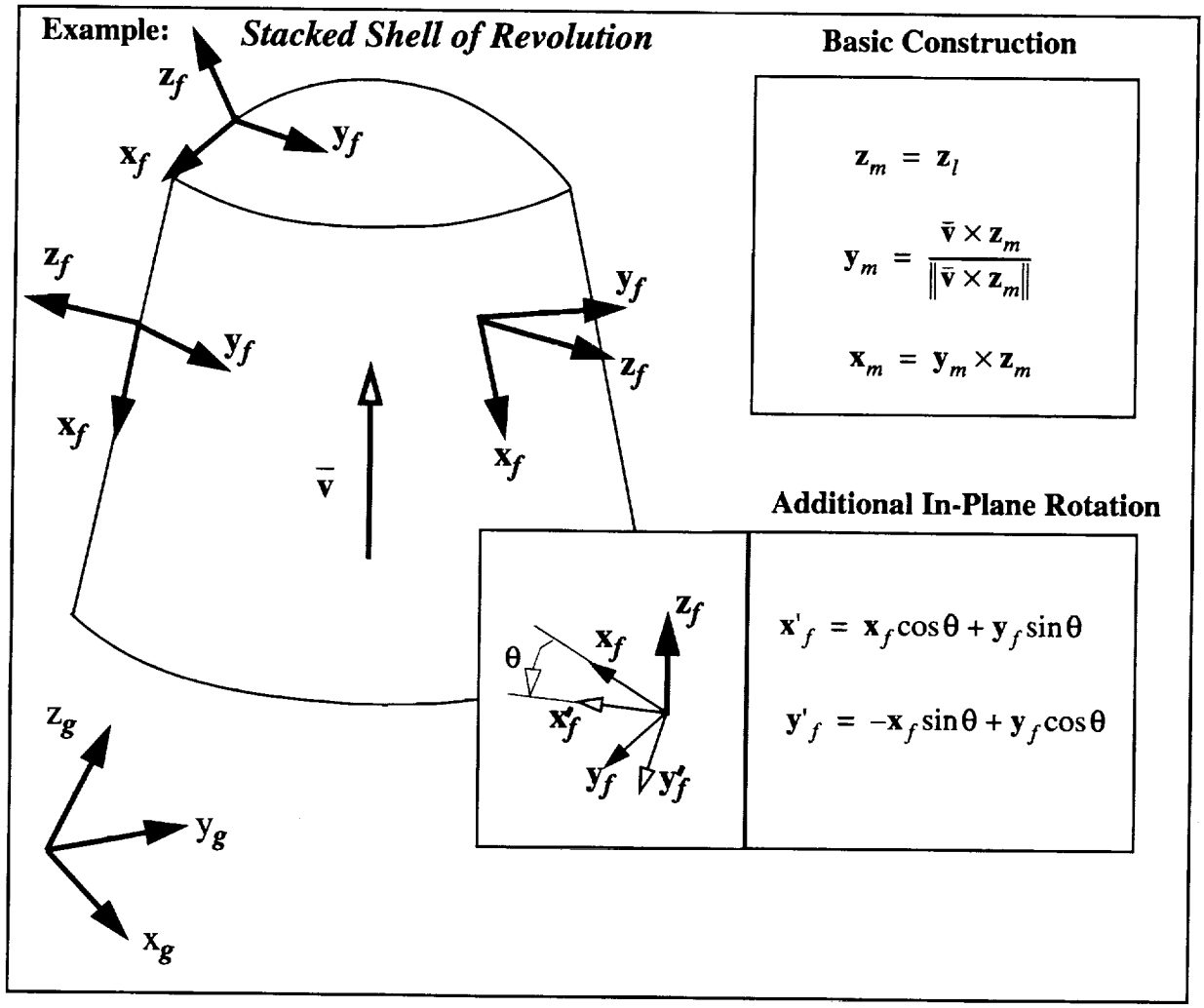


Figure 2.7-4 FAB_DIR = VECTOR Option

2.7.6 The FAB_DIR = PLANE Option

The PLANE option is invoked by issuing the DEFINE ELEMENTS subcommand.

FAB_DIR = PLANE $\bar{u}, \bar{v}$ [, θ] [/3D]

where $\bar{u}$ and $\bar{v}$ denote the global components of two arbitrary reference vectors, i.e., $\bar{u} = \bar{u}_{xg}, \bar{u}_{yg}, \bar{u}_{zg}$, and $\bar{v} = \bar{v}_{xg}, \bar{v}_{yg}, \bar{v}_{zg}$, which together represent a plane in 3D space. Tentatively, the x_f and y_f directions are located in this plane with z_f normal to it. For shells, this tentative triad is then rotated into the element tangent plane (at each integration point) by projecting the initial z_f axis into the local element normal (z_l) axis. As with the POINT and VECTOR options, an optional in-plane rotation of θ may be performed about the z_f direction to obtain the final orientation of the fabrication frame. The PLANE option may be used to generate fabrication frame triads for general shell and 3D solid structures. If the /3D qualifier is used, the normal projection step is bypassed. The construction is illustrated in Figure 2.7-5.

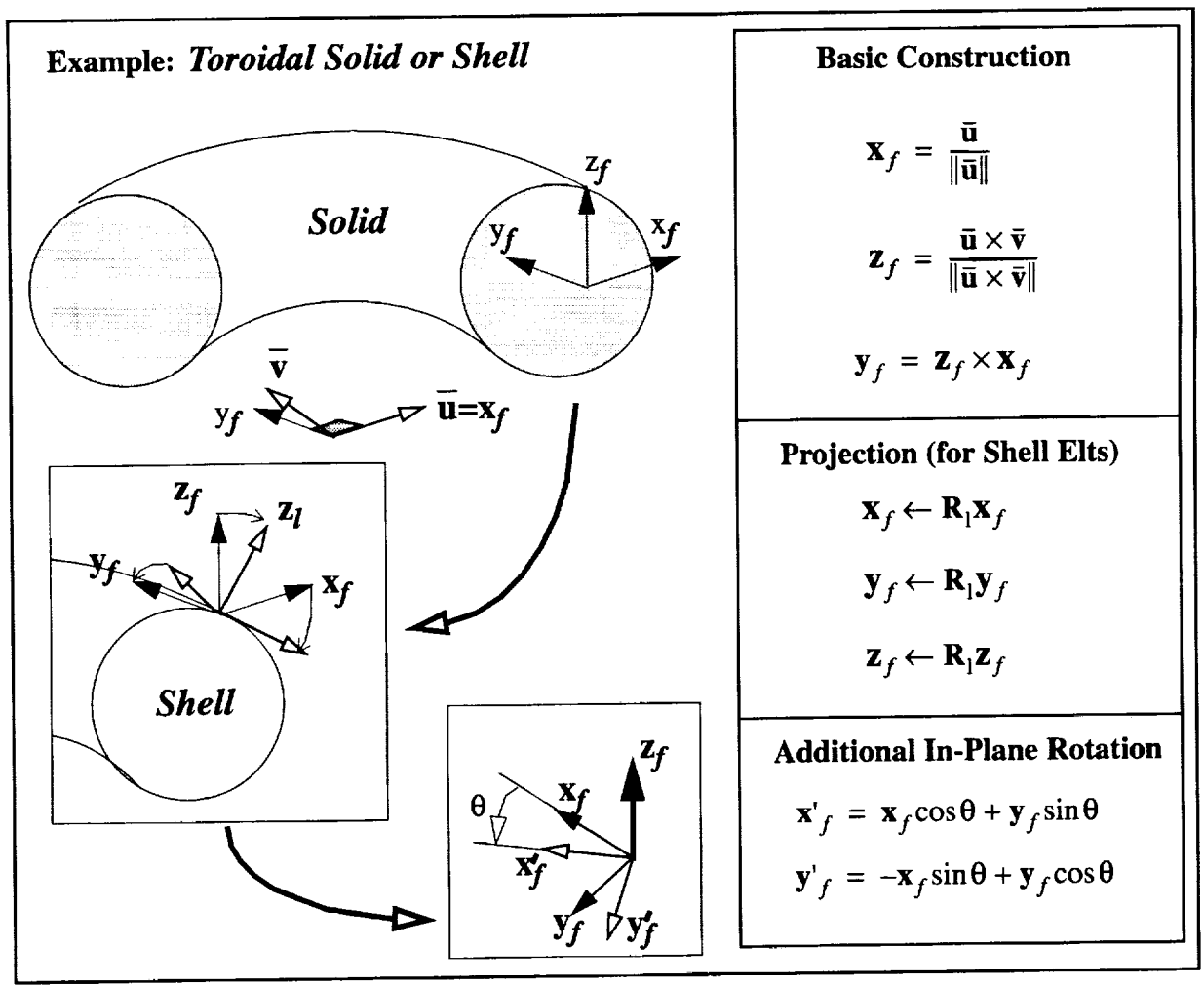


Figure 2.7-5 FAB_DIR = PLANE Option

2.7.7 The FAB_DIR = BEAM Option

The BEAM option is invoked by issuing the DEFINE ELEMENTS subcommand.

FAB_DIR = BEAM *node*

where *node* denotes the node number of a beam element reference point, with coordinates, $\bar{x} = \bar{x}_g$, $\bar{y}_g$, $\bar{z}_g$, which in conjunction with the two end points (i.e., nodes) of a beam element define the element's corotational frame (x_e, y_e, z_e). For straight beam elements, the fabrication/cross-section frame (x_f, y_f, z_f) is then coincident with the corotational frame, as is the element stress frame (x_l, y_l, z_l). For curved beam elements, the reference point must be defined in the same plane as the first three beam element nodes, and only the z axes of the element corotational and fabrication/stress frames will coincide, as illustrated in Figure 2.7-6.

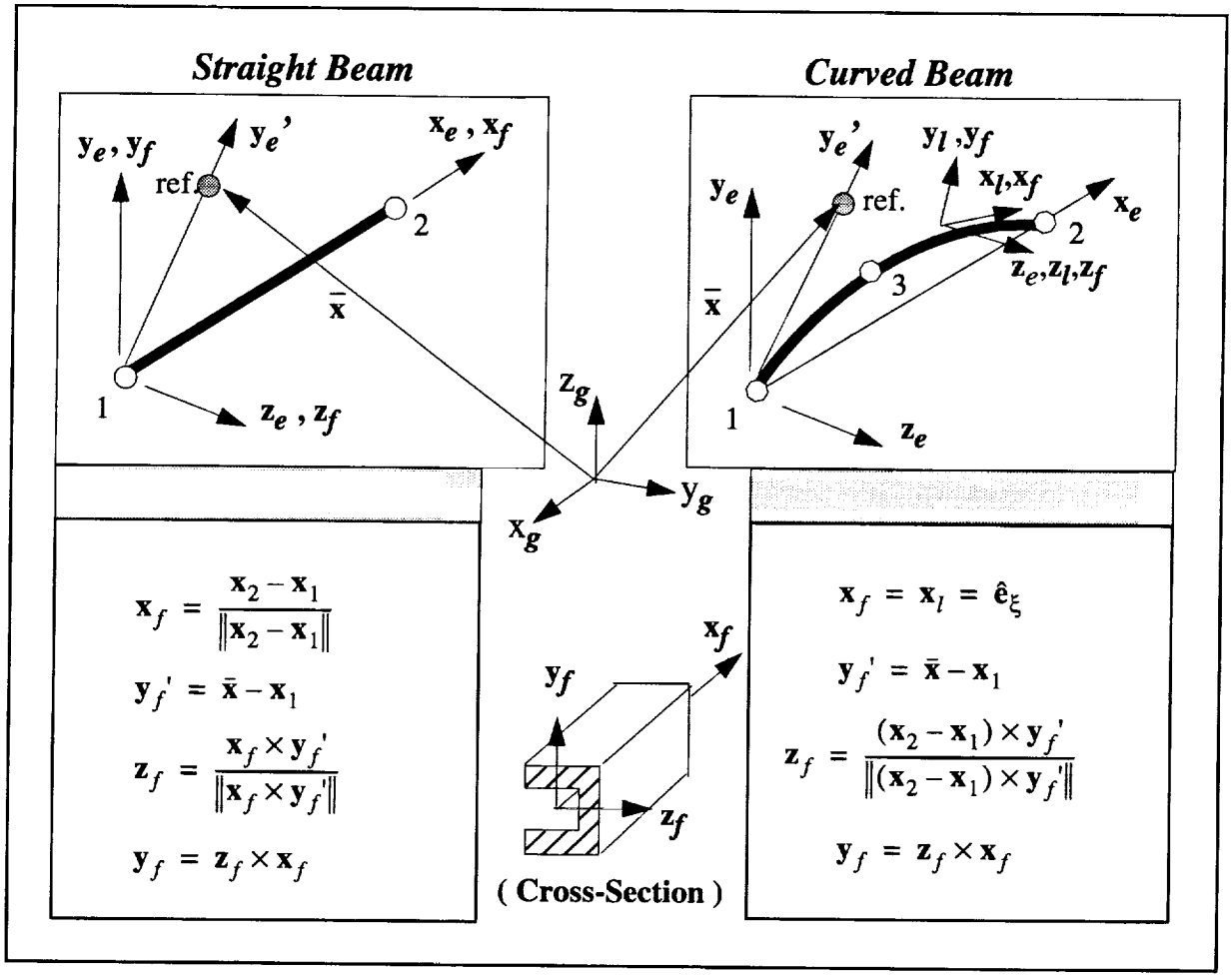


Figure 2.7-6 FAB_DIR = BEAM Option

2.8 Load Definition Procedures

2.8.1 General Description

The Load Definition part of a Model Definition Procedure is described in this section. Load Definition includes the definition of element (distributed) and/or nodal (concentrated) loads. Element loads are defined via the Generic Element Processor (i.e., Processors ES*i*). Nodal loads are defined via Processor AUS, which builds a simple table in the old (COMET-BL) format, and Processor REDO, which converts the AUS table to a new dataset compatible with the COMET-AR database. The commands for load definition can either be added directly to the Model Definition Procedure or placed in a sub-procedure as illustrated in Box 2.8-1.

Box 2.8-1 Sample Load Definition Sub-Procedure

```
*PROCEDURE LoadDefn ( load_arg_1 [ = model_val_j1 ]; ... load_arg_n [ = model_val_jn ] )
   . Define Element (Distributed) Loads for Selected Element Processors/Types/Locations
      RUN ESi
         RESET ELEMENT_TYPE = EltTyp
         DEFINE LOADS /TYPE=LoadTyp
            ELEMENTS=elt_range ; Boundaries=bndy_range ; NODE=nod_range
               LOAD = load values
            :::
         END_DEFINE_LOADS
      :::
      STOP
   . Define Nodal (Concentrated) Loads for Selected Nodes and DOFs
      RUN AUS
         SYSVEC : APPL LoadName 1 1 . ( LoadNam e= FORCE or DISP )
            i = dof_number_1 : j = node_number_1 : load_value_1
            i = dof_number_2 : j = node_number_2 : load_value_2
            :
      RUN REDO
         NVT APPL.LoadName.1.1 NODAL.SPEC_LoadName.1
      STOP
*END
```

In the above procedure (arbitrarily called *LoadDefn*), a subset of Model Definition Procedure argument values have been transferred from above (i.e., *model_arg_j1 ... model_arg_jn*). These values are now referred to with the local arguments *load_arg_1 ... load_arg_n* and may be employed within the Load Definition Procedure by using square brackets for symbolic replacement (e.g., [*load_arg_1*]). These arguments typically are used to pass load magnitude and/or type parameters from the main Model Definition procedure.

For element (distributed) load definitions, a separate element processor (ES*i*) must be RUN for each element type that is to be loaded. Only those element types that have been employed within the Element Definition Procedure (see previous section) are relevant here. After specifying the element type within the processor (via the RESET ELEMENT_TYPE command) a DEFINE LOADS command is needed for each separate load type that is to be applied. Element load types refer to pressure, line loads, body loads, and temperatures, as summarized in the following subsection. They are stored in the database as distributed loads (i.e., per unit length, area or mass) and during the solution phase are converted into consistent nodal forces by the element processor(s). In Box 2.8-1, the user may specify a selected range of elements (and/or groups), element boundaries, and element boundary nodes, before specifying the load values via a LOAD command. For details on the DEFINE LOADS command, refer to Section 7.2, *Generic Element Processor.*

For nodal (concentrated) load definition, processor AUS is used to construct a table of specified nodal force values and/or a table of specified nodal displacement values. Each of these tables contains a column for each node in the model, and a row for each nodal DOF in the model (e.g., six rows for shell element models). Only those nodal DOFs that are to be loaded are mentioned in the AUS command stream. Unspecified nodal DOFs are assumed to be unloaded (i.e., unspecified forces are assumed to be zero, and unspecified displacements are assumed to be free unless constrained by boundary conditions. Processor REDO must be executed after creating specified nodal force and/or displacement tables with AUS to convert these tables to the standard COMET-AR nodal vector dataset format (via the NVT command). See Section 6.7 on Processor REDO for details.

> Specified nodal displacements are relevant only for nodal DOFs that are designated SPCnz (i.e., specified nonzero) during boundary condition definition (see Section 2.9, *Boundary Condition Definition Procedures*).

> Specified nodal forces are not recommended for use with adaptive mesh refinement. Such concentrated forces can lead to singularities in the solution, and should be replaced where possible by local element distributed loads. In contrast, specified nodal displacement are fine for adaptive analysis, as they can simply be interpolated when attached elements are subdivided by the adaptive algorithm.

2.8.2 Available Load Types in COMET-AR

A summary of element load types currently available in COMET-AR is given in Table 2.8-1.

Table 2.8-1 Summary of COMET-AR Element (Distributed) Load Types

Load Type	Description
LINE	Force and/or moment vectors per unit length; specified at nodes on selected element edges.
PRESSURE	Normal force per unit area; specified at nodes on selected element surfaces; may be "dead" (fixed direction) or "live" (follower) force.
SURFACE	General traction vectors (force and/or moment per unit area) specified at nodes on selected element surfaces.
BODY	Body force vector per unit mass; specified at element nodes.
TEMP	Thermal loads; temperature values specified at element nodes. (Currently untested)

Detailed instructions for defining each of the above element load types may be found in Section 7.2, *Generic Element Processor.* Each of the above element load types is specified at element nodes and interpolated along an element line, surface, or volume (depending on the load type) via the element's intrinsic shape functions.

> Not all element processors/types have all of the above load types implemented. Check the subsection on Element Processor Limitations under the appropriate ES*i* processor section in Chapter 7 for specific element-load status information.

Nodal (concentrated) load types are summarized in Table 2.8-2.

Table 2.8-2 Summary of COMET-AR Nodal (Concentrated) Load Types

Load Type	Description
FORCE	Concentrated forces and/or moments at selected nodal DOFs.
DISPLACEMENT	Concentrated displacements (translations and/or rotations) at selected nodal DOFs.

2.9 Boundary Condition Definition Procedures

2.9.1 General Description

The Boundary Definition part of a Model Definition Procedure is described in this section. Boundary Condition Definition includes the designation of active (free) and inactive (suppressed or specified non-zero) nodal DOFs, as well as the definition of multi-point constraints (MPCs) which constrain selected nodal DOFs to be linear combinations of other nodal DOFs. All such boundary conditions are defined via Processor COP (the Constraint Processor), which is described in Section 6.2. The COP commands for boundary condition definition can either be added directly to the Model Definition Procedure or placed in a sub-procedure as illustrated in Box 2.9-1.

Box 2.9-1 Sample Boundary Condition Definition Sub-Procedure

***PROCEDURE BCsDefn** (*bcs_arg_1* = *bcs_def_1* ; . . . *bcs_arg_n* =*bcs_def_n*)

. Run Constraint Processor to Define all Boundary Conditions

RUN COP

MODEL

SELECT NEW DOFDAT *ldi, conset, mesh* . (e.g., 1, 1, 0)

CONSTRAIN

. Designate suppressed (specified zero) nodal DOFs

ZERO NODE = $node1_1$, $node2_1$, $nodeinc_1$ DOF = $dofnam1_1$, $dofnam2_1$, ...

ZERO NODE = $node1_2$, $node2_2$, $nodeinc_2$ DOF = $dofnam1_2$, $dofnam2_2$, ...

:::

. Designate specified non-zero nodal DOFs

NONZERO NODE = $node1^1$, $node2^1$, $nodeinc^1$ DOF = $dofnam1^1$, $dofnam2^1$, ...

NONZERO NODE = $node1^1$, $node2^1$, $nodeinc^1$ DOF = $dofnam1^1$, $dofnam2^1$, ...

:::

. Define Multi-Point/DOF Constraints

MPC :::

:::

DONE

STOP

*END

In the procedure shown in Box 2.9-1 (arbitrarily called *BCsDefn*), a subset of Model Definition Procedure argument values have been transferred from above into the local arguments *bcs_arg_1 ... bcs_arg_n* and may be employed within the Boundary Condition Definition Procedure by using square brackets for symbolic replacement (e.g., [*bcs_arg_1*]). These arguments are typically used to pass boundary condition option parameters from the main Model Definition procedure.

After running the Constraint Processor (COP), the MODEL and SELECT commands are used to create a nodal DOF dataset, *NODAL.DOF..conset.mesh*, on the database file connected to logical device index *ldi*. The CONSTRAIN command then initiates the definition of specified zero and nonzero nodal DOFs, via the ZERO and NONZERO subcommands, respectively. In these subcommands, *node1*, *node2*, and *nodeinc* represent a range (first, last, and increment) of global node numbers, and *dofnami* represents a valid DOF name (e.g., d1, d2, d3, theta1, theta2, or theta3). The MPC subcommand is used to define any multipoint constraints present. Finally, the DONE command is used to terminate the constraint (boundary condition) definition and the STOP command is used to terminate processor COP.

2.9.2 Available Boundary Condition and DOF Types in COMET-AR

A summary of boundary condition types now available in COMET-AR is given in Table 2.9-1. All boundary conditions refer to nodal DOFs. There are no element, edge, or surface-oriented boundary conditions except as created by the user in the Boundary Condition Procedure.

Table 2.9-1 Summary of COMET-AR Nodal Boundary Condition Types

BC Type	Description
ZERO (or SPCz)	Nodal DOFs that are totally suppressed. These may be specified via Processor COP's ZERO subcommand, or generated automatically via the Automatic DOF Suppression option discussed in the next section.
NONZERO (or SPCnz)	Nodal DOFs that are set to some prescribed value by the user. The node and DOF numbers should be specified via Processor COP's NONZERO subcommand. The actual prescribed (base) values should be set in the Load Definition Procedure, via Processor AUS (see the previous section).
MPC (Multi-Point Constraint)	Nodal DOFs that are expressed as a linear combination of other nodal DOFs (either at the same or at different nodes). These dependent DOFs are later eliminated from the equation system through an assembly transformation (see Processor ASM). The node/DOF numbers and the coefficients appearing in the linear constraint may all be specified by the user via Processor COP's MPC subcommand.
FREE (or Active)	Nodal DOFs that are neither specified as zero or nonzero and which do not appear as a dependent variable in a multipoint constraint, are considered free and constitute unknowns in the assembled equation system.

A summary of nodal DOF types currently recognized by COMET-AR is given in Table 2.9-2.

Table 2.9-2 Summary of COMET-AR Potential Nodal DOF Types

DOF Name	Description
d1, d2, d3	Translational displacements in the Computational Frame's x_c, y_c, z_c directions, respectively.
theta1, theta2, theta3	Rotational displacement about the Computational Frame's x_c, y_c, z_c axes, respectively.

Detailed instructions for associating any of the boundary condition types listed in Table 2.9-1 with any of the nodal DOF types listed in Table 2.9-2 may be found in Section 6.3 on Processor COP.

The default set of DOFs at all nodes is 6, i.e., 3 translations and 3 rotations: d1, d2, d3, theta1, theta2, theta3. While COP provides a special command to change or expand/reduce this default DOF pattern, the non-default options have not been sufficiently tested in COMET-AR.

Any nodal DOFs that are not mentioned in a ZERO or NONZERO COP subcommand are assumed to be free (i.e., active). Processor COP also has a FREE command to release any nodal DOFs that have been unintentionally constrained by previous ZERO or NONZERO commands.

Multi-point constraints (MPCs) in COMET-AR are currently restricted to be linear and explicit. There must be a clear distinction between dependent and independent DOFs appearing in a linear constraint equation, so that the (one) dependent DOF in a given constraint equation can be eliminated from the assembled equation system (as opposed to the use of Lagrange multipliers or penalty methods, which add DOFs or stiffness to the equation system). If there are any multi-point constraints present, the user must select Processor ASM as the assembly processor option when invoking a Solution Procedure.

During adaptive mesh refinement (AR), new nodes are automatically generated by the refinement processor (e.g., REF1) and appropriate boundary conditions for each new node are deduced from the boundary condition types associated with neighboring nodes on attached element boundaries. Unless a user-written solid model interface is employed (see Chapter 16) this boundary condition deduction process is not fool-proof. For the time being, the user should monitor the constraints assigned to AR-generated nodal DOFs via the COMET-AR graphical post-processor, ARGx.

In addition to basic boundary condition definition, COMET-AR provides some automatic DOF suppression options to eliminate unstable nodal DOFs, i.e., nodal DOFs that are not supported by element stiffness such as shell drilling rotations, or rotations in general at nodes connected only to solid elements. For more information on this capability, see Section 2.10, *Automatic DOF Suppression and Drilling Stabilization*.

Nodal DOFs that are constrained to be either zero, non-zero, or a linear combination of other DOFs (i.e., MPC) may or not have an equation number assigned to them in the assembled matrix equation system. The decision as to which option to employ is typically made internally, within particular COMET-AR solution and/or utility procedures.

2.10 Automatic DOF Suppression and Drilling Stabilization

The model boundary conditions defined by the user may not be sufficient to remove all extraneous DOFs (i.e., DOFs for which there is negligible element stiffness present). Then the assembled equation system may be nearly (or completely) singular, hence unsolvable. To avoid this pitfall, COMET-AR provides an automatic DOF suppression capability (AUTO_DOF_SUP) for the general situation, and two special-purpose options (AUTO_DRILL and AUTO_TRIAD) for treating extraneous drilling rotational DOFs associated with shell elements that may be missed by the AUTO_DOF option. The selection of one or more of these options is not made by the user until the solution phase, and AUTO_DOF_SUP, AUTO_DRILL and AUTO_TRIAD appear as solution procedure arguments.

2.10.1 Basic Automatic DOF Suppression Option (AUTO_DOF_SUP)

The basic automatic DOF suppression option, AUTO_DOF_SUP, suppresses all nodal DOFs that do not have sufficient element stiffness in the corresponding computational directions. For example, all rotational DOFs may be suppressed at nodes that are connected only to solid elements (which typically have only translational stiffness); selected drilling rotational DOFs may be suppressed at nodes connected to shell elements if the element normal vectors are sufficiently close to one of the computational axes at the node (see Figure 2.10-1).

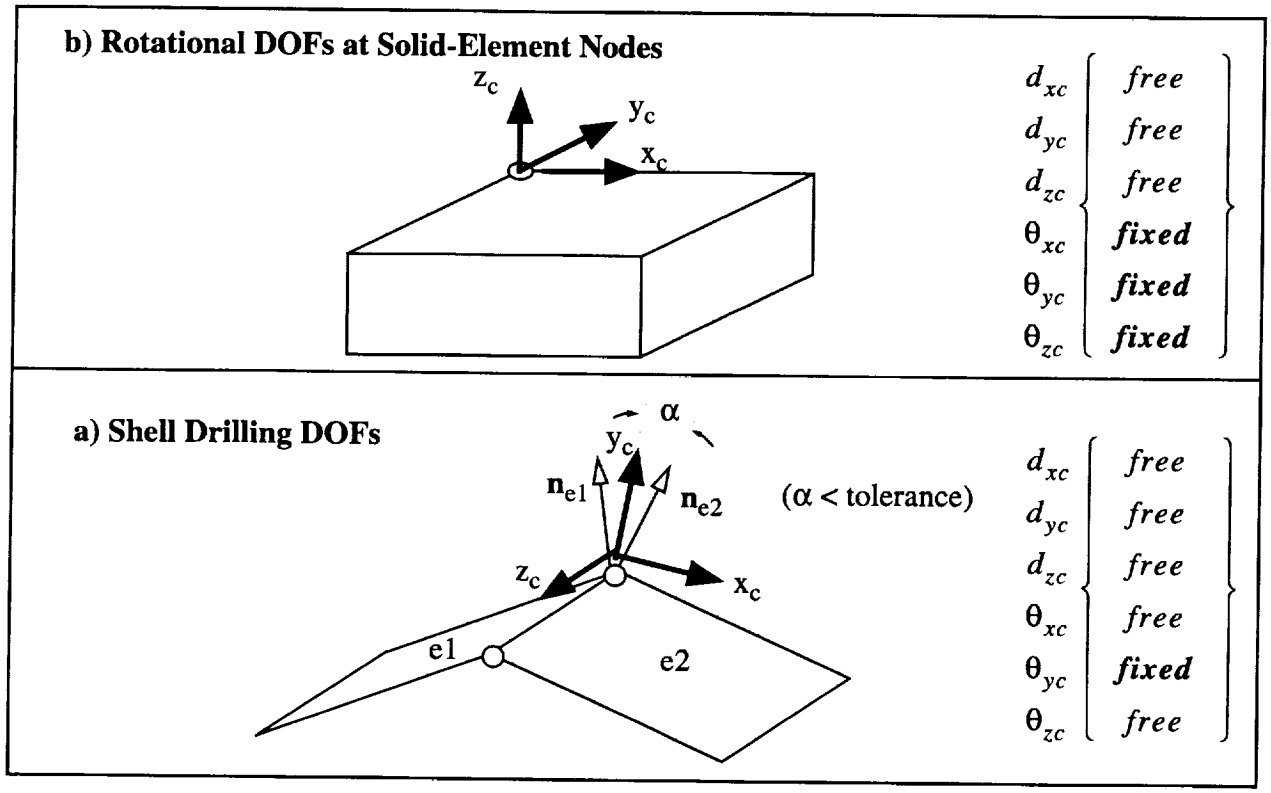


Figure 2.10-1 Examples of DOFs Suppressed by AUTO_DOF_SUP Option

The basic steps involved in automatic DOF suppression via the AUTO_DOF_SUP option are summarized in Table 2.10-1. The user performs the first two steps; COMET-AR does the rest.

Table 2.10-1 Steps in AUTO_DOF_SUP Algorithm

Step	Description
1	The user defines all physical boundary conditions for the model, as described in the section on Boundary Condition Definition Procedures. This leads to the creation of a NODAL.DOF dataset with nodal DOF boundary condition types set to FREE, ZERO, NONZERO, or MPC.
2	The user selects the AUTO_DOF_SUP option from one of the COMET-AR Solution Procedures (e.g., L_STATIC_1 or AR_CONTROL).
3	The solution procedure creates an auxiliary nodal DOF dataset, which is called ELT_NODAL.DOF, for elements to indicate which nodal DOFs they support with stiffness. This dataset is initialized such that all nodal DOFs are set to SPCz (i.e., suppressed).
4	The solution procedure executes all relevant element processors, and for each element, nodal DOFs that have stiffness in one of the computational directions are switched to FREE in the ELT_NODAL.DOF dataset. If there is no stiffness contribution from the element, the nodal DOF setting is left as-is.
5	After processing all elements, the ELT_NODAL.DOF dataset reflects a setting of FREE for all nodal DOFs that have supporting element stiffness, and SPCz for all nodal DOFs that have negligible stiffness.
6	The ELT_NODAL.DOF dataset is then merged with the NODAL.DOF dataset, so that all nodal DOFs that are set to SPCz in the ELT_NODAL.DOF dataset are also set to SPCz (i.e., suppressed) in the NODAL.DOF dataset.
7	The resulting NODAL.DOF dataset contains all of the user's original boundary condition assignments plus any extra DOF suppressions contributed from the ELT_NODAL.DOF dataset. Superfluous nodal DOFs have been automatically suppressed.

2.10.2 Stabilization of Drilling DOFs (AUTO_DRILL/AUTO_TRIAD/AUTO_MPC)

2.10.2.1 General Description

Many of the shell elements in COMET-AR intrinsically have only 5 DOFs per node: 3 translations and 2 rotations. The 3rd, or "drilling," rotational DOF, which is a rotation about the shell element normal direction, does not appear in the shell theory and thus has no intrinsic stiffness associated with it. This rank-deficiency can lead to singularities in the assembled stiffness matrix, preventing a solution of the equation system with conventional equation solvers. In some cases, the problem can be easily remedied, such as when the Computational Frame at each node is defined such that one of the computational axes (x_c, y_c, or z_c) is nearly aligned with the element nodal normal and the drilling DOF can be suppressed a priori (e.g., via the AUTO_DOF_SUP option described above). At nodes where shell elements intersect at sufficiently large angles, rank-deficiency is avoided without having to suppress any DOFs, as the drilling rotation in one element is resisted by the bending stiffness in the adjacent element.

For more general situations (see Figure 2.10-2) where smooth shell regions exist in which the computational axes can not be conveniently aligned with the element drilling rotation, additional measures are necessary. Three mutually exclusive options are available within COMET-AR:

1) **AUTO_DRILL**: the addition of artificial drilling stiffness at the element level (for certain element processors); or
2) **AUTO_TRIAD**: automatic re-direction of the computational axes so that the drilling rotation can be suppressed afterwards by the AUTO_DOF_SUP option; or
3) **AUTO_MPC**: automatic generation of multipoint constraints (MPC—actually multi-DOF constraints) at a point to suppress drilling DOFS regardless of the directions of the computational axes.

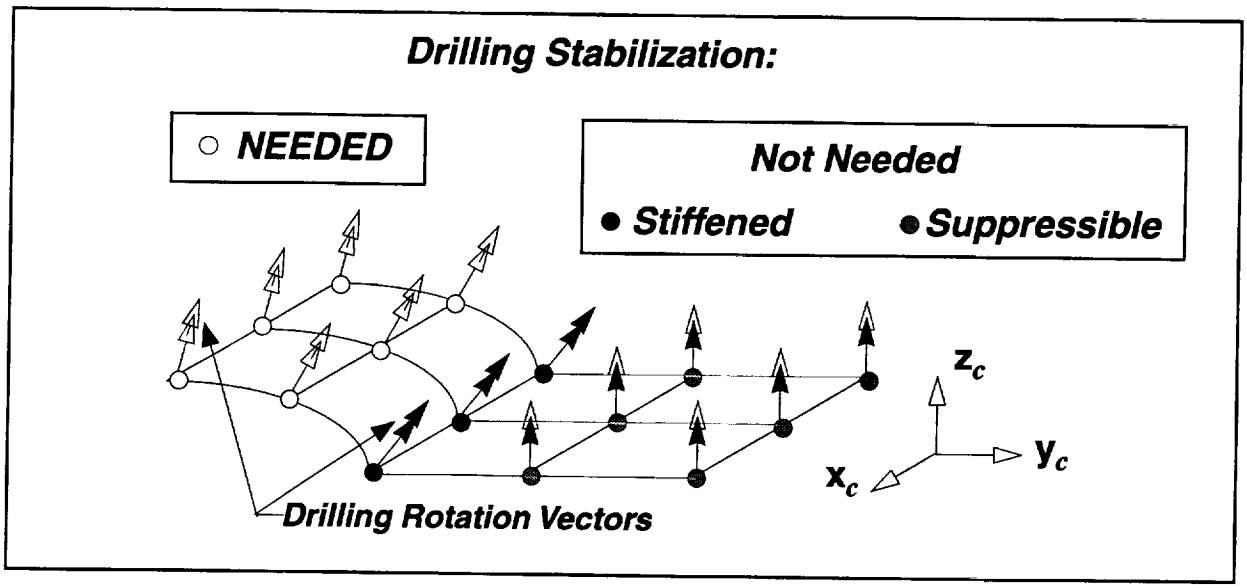


Figure 2.10-2 Motivation for Drilling DOF Stabilization

2.10.2.2 Automatic Drilling Stiffness Option (AUTO_DRILL)

Artificial drilling stiffness is available in most COMET-AR shell element processors that do not have intrinsic drilling stiffness (e.g., ES1p and ES7p). In these processors the addition of artificial drilling stiffness is triggered via the AUTO_DRILL solution procedure argument. This option insures that drilling stiffness is added at the element level, but only where needed.

For built-up shell structures, it is neither necessary nor desirable to add artificial drilling rotational stiffness at nodes where elements intersect at moderately large angles, e.g., along the panel/stiffener juncture line in a blade-stiffened panel (see Figure 2.10-3). At such nodes, sufficient rotational stiffness is already provided in all three computational directions by the assembly of bending stiffnesses from the contributing adjacent elements. If the ratio of the thicknesses for the intersecting elements is large, the addition of artificial drilling stiffness from the thicker element may overwhelm the bending stiffness in the attached element and adversely affect accuracy. The

AUTO_DRILL option thus turns nodal drilling stiffness flags on selectively, based on whether the structure is smooth or junctured at each node, as illustrated in Figure 2.10-3.

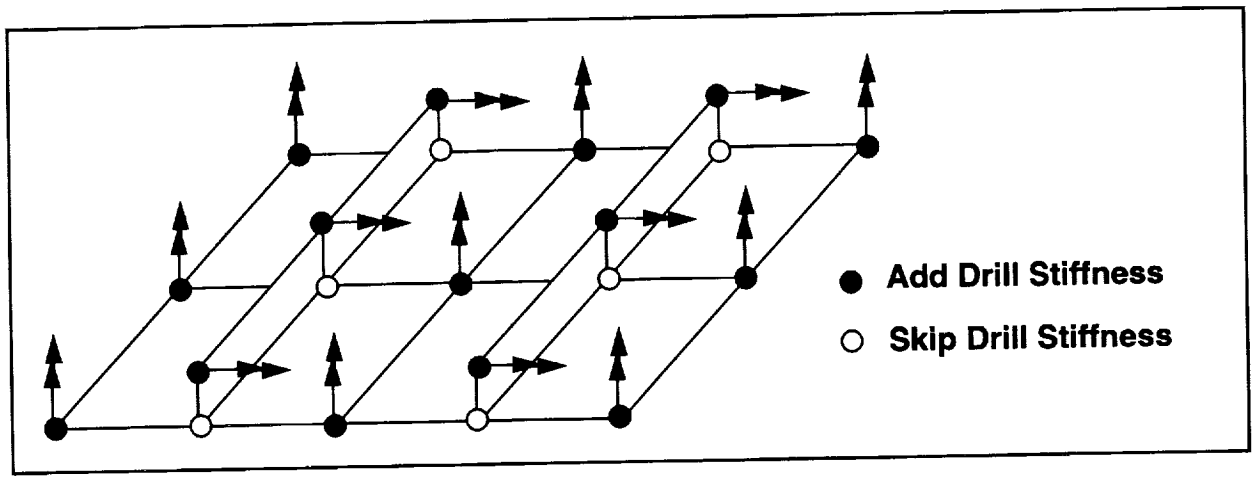


Figure 2.10-3 Effect of Automatic Drilling Stiffness Selection Option

When the AUTO_DRILL option is selected by the user (at the solution procedure level) two things happen: 1) artificial drilling stiffness flags are defined for each node in the model, indicating where drilling stiffness is needed; and 2) during element stiffness formation, shell elements attached to nodes that are flagged for drilling stiffness add a "small" diagonal stiffness contribution to the normal rotational component at those nodes. An optional drilling stiffness magnitude parameter and an optional drilling stiffness angle tolerance parameter are provided in conjunction with the AUTO_DRILL argument appearing in COMET-AR Solution Procedures.

> Some element processors, such as ES36, have artificial drilling stiffness hardwired in the element formulation. For such elements, the AUTO_DRILL option is irrelevant, as the drilling stiffness is added at the element level whether or not the option is selected.

2.10.2.3 The AUTO_TRIAD Option

The AUTO_TRIAD option is an alternative to AUTO_DRILL that bypasses the need for artificial drilling stiffness and some of the numerical difficulties associated with it (especially in nonlinear analysis). With this option, computational triads (x_c, y_c, z_c) are re-oriented at all nodes not subject to boundary conditions, such that one of the computational axes is aligned with the average element normal at the node. The effect of AUTO_TRIAD is illustrated by example in Figure 2.10-4. At the "black" node the computational triad $\{x_c, y_c, z_c\}$ is originally aligned with the global triad $\{x_g, y_g, z_g\}$. The AUTO_TRIAD option then replaces that triad with the new triad $\{\bar{x}_c, \bar{y}_c, \bar{z}_c\}$ such that the new $\bar{z}_c$ axis is aligned with the average element normal at the node, which in the figure is close enough to the individual element normals that the "drilling" rotation about the z_c axis can be suppressed automatically by the AUTO_DOF_SUP option.

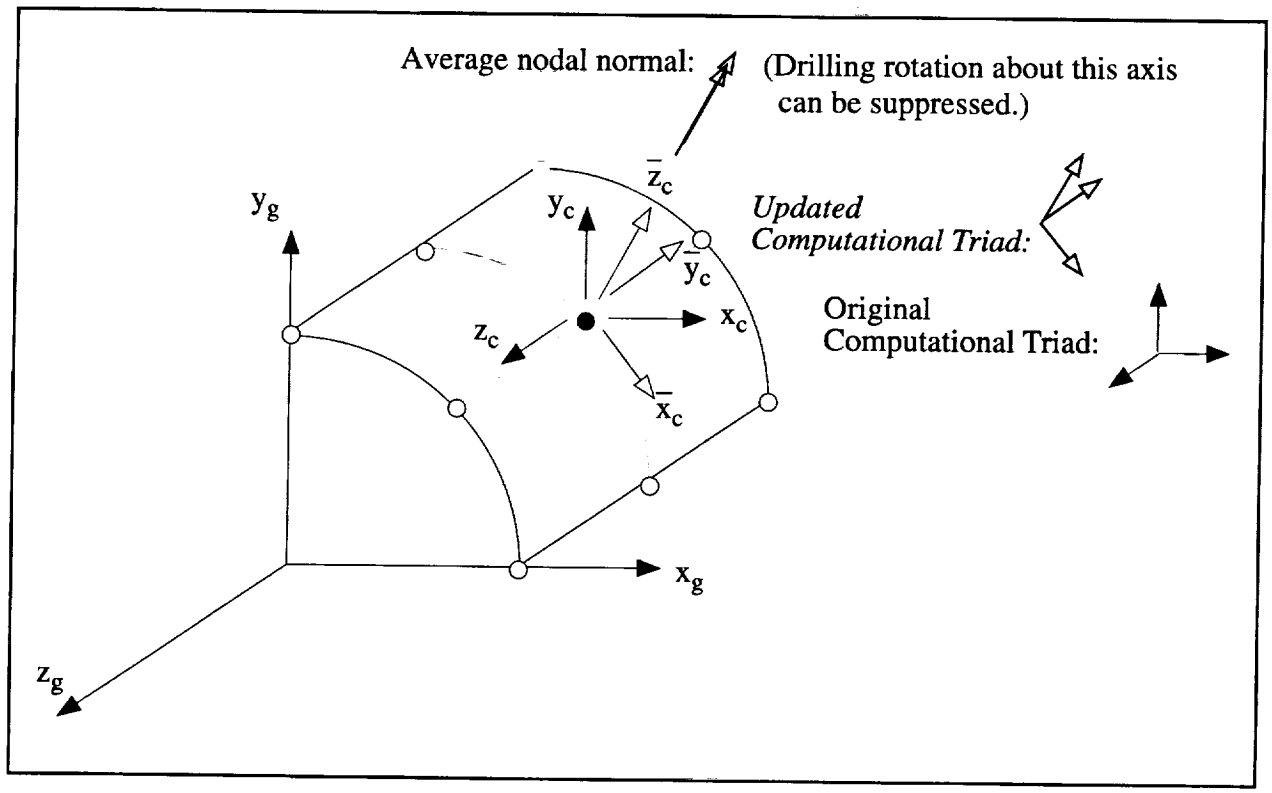


Figure 2.10-4 Illustration of AUTO_TRIAD Option at a Node

The AUTO_TRIAD option must be used in conjunction with the AUTO_DOF_SUP option, so that the corresponding drilling rotational DOFs are automatically suppressed at all nodes where insufficient drilling stiffness exists.

Computational triads at nodes with any DOFs assigned boundary conditions (e.g., suppressed or specified nonzero) are skipped by the AUTO_TRIAD option. The user is responsible for stabilizing drilling DOFs at these nodes.

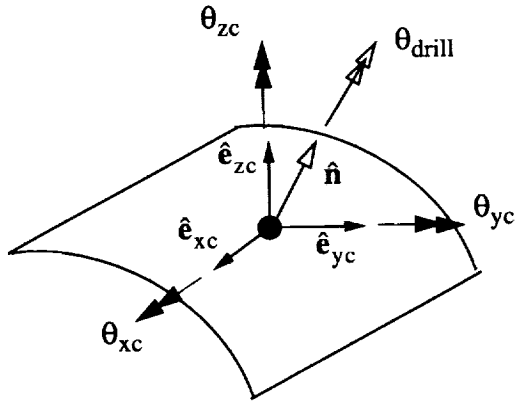
Concentrated nodal forces (or moments) should not be employed in conjunction with the AUTO_TRIAD option, as the computational triads may be inadvertently re-directed by the program, changing the interpretation of the force components. User-defined multi-point constraints should also not be used in conjunction with the AUTO_TRIAD option for the same reasons.

When post-processing displacement results obtained with the AUTO_TRIAD option, remember that nodal displacements will be expressed with respect to the re-directed computational axes and may need to be transformed back to the global frame. These transformations are automatically performed by COMET-AR post-processors, such as ARGx.

2.10.2.4 The AUTO_MPC Option

The AUTO_MPC option is the most direct and robust way to eliminate unstable drilling rotational DOFs. It automatically generates an explicit, multi-DOF constraint equation, suppressing the drilling rotation for each node where there is insufficient stiffness to stabilize (i.e., resist) that particular motion. The nature of the constraint equation is shown both geometrically and algebraically in Figure 2.10-5. Here, $\hat{e}_{xc}$, $\hat{e}_{yc}$, $\hat{e}_{zc}$ represent unit vectors in three mutually perpendicular computational directions: x_c, y_c, z_c, respectively; θ_{xc}, θ_{yc}, θ_{zc} represent the corresponding rotational DOFS at the node; $\hat{n}$ represents the average unit normal vector at the node (i.e., the drilling direction); and θ_{drill} represents the corresponding drilling rotation. The drilling constraint involves all three rotational DOFS about the computational axes (x_c, y_c, z_c), and the computational frame at such nodes may be totally arbitrary, with both loads and boundary conditions present as well. This is in contrast to the AUTO_TRIAD option, where the computational frames are automatically modified by the code; or to the basic AUTO_DOF_SUP option where the user is responsible for properly aligning one of the computational axes with the drilling axis at nodes that do not lie on shell/shell or shell/beam juncture lines.

The AUTO_MPC option is the most all-purpose and robust of the drilling stabilization options. The only disadvantage of the AUTO_MPC option over other AUTO options is that is newer and is less tested.



Constraint Equation: $\theta_{drill} = \theta_{xc}(\hat{n} \cdot \mathbf{e}_{xc}) + \theta_{yc}(\hat{n} \cdot \mathbf{e}_{yc}) + \theta_{zc}(\hat{n} \cdot \mathbf{e}_{zc}) = 0$

Figure 2.10-5 The AUTO_MPC Option for Stabilizing Drilling Rotations

2.10.3 Usage Guidelines/Limitations for AUTO_DOF/DRILL/MPC/TRIAD

The guidelines/limitations listed in Table 2.10-2 should be considered when selecting any of the above options at the solution procedure level and when defining the model.

Table 2.10-2 Usage Guidelines for AUTO_DOF, AUTO_DRILL , AUTO_MPC, and AUTO_TRIAD Options

Number	Guideline
1	AUTO_DOF_SUP should always be selected as a safeguard
2	AUTO_MPC is the recommended option for ensuring that unstable drilling DOFs are properly suppressed, for both linear and nonlinear, adaptive and non-adaptive analysis. Proviso: While the AUTO_MPC approach is in principle the most robust, the software is newer than the other AUTO options and hence may still have some bugs.
3	AUTO_DRILL should be selected only for linear analysis, and not in conjunction with iterative equation solvers.
4	AUTO_MPC or AUTO_TRIAD are alternatives to AUTO_DRILL for nonlinear analysis.
5	AUTO_TRIAD must be used in conjunction with AUTO_DOF_SUP.
6	AUTO_TRIAD should not be used if concentrated nodal forces are present.
7	AUTO_TRIAD does not process nodes for which any DOFs have been assigned boundary conditions; the user is responsible for drilling DOF suppression at such nodes.
8	If the computational frames align closely with the average shell-element normals throughout the model, then neither AUTO_DRILL nor AUTO_TRIAD is necessary, only AUTO_DOF_SUP.

Details on the parameters available with the AUTO_DOF, AUTO_DRILL, AUTO_MPC, and AUTO_TRIAD options are provided under the description of the solution procedures in which they appear as arguments (see, e.g., L_STATIC_1 or AR_CONTROL).

2.11 Sample Model Definition Procedures (Summary)

A number of existing model definition procedures, listed in Table 2.11-1, are available for the interested reader to peruse (or cannibalize) on the computer. These procedures have been developed during the course of research on adaptive finite element methods sponsored by NASA Langley Research Center. They range from extremely simple geometries, such as an L-shaped domain, to moderately simple geometries, such as an I-stiffened panel. For the definition of more complicated models (such an aircraft structure) it is advisable for the user to employ an automatic mesh generation package, such as PATRAN, in conjunction with the PATRAN-to-COMET-AR converter (see Section 6.6) rather than manually write a command-language procedure such as those described in the preceding sections.

Table 2.11-1 Some Existing COMET-AR Model Definition Procedures

File Name	Model Name	Description
bsp.clp	Blade-Stiffened Panel	Flat plate with 4 axial blade stiffeners
bspx.clp	Cut Blade-Stiffened Panel	Same as bsp, but with one cut-off stiffener
crp.clp	Cracked Plate	Flat plate with partial crack
fkp.clp	Flat "Knight's Panel"	Flat version of kp (panel with circular hole)
isp.clp	I-Stiffened Panel	Flat/curved panel with 4 "T" stiffeners
kp.clp	Knight's Panel	Composite cylindrical panel with circular hole
lsd.clp	L-Shaped Domain	Flat plate with square cutout; 1/4 model
pc.clp	Pinched Cylinder	Cylindrical shell with opposing point/line loads
pwh.clp	Plate with Hole	Flat plate with circular hole, under tension
scb.clp	Short Cantilevered Beam	Rectangular plate, clamped at one end
steele_cyl.clp	Steele's Cylinder	Axisymmetric model of cylindrical shell
steele_tor.clp	Steele's Toroid	Axisymmetric model of toroidal shell

The above model definition files may be found on the computer in the directory:

comet-ar-root **/prc/applications**

where *comet-ar-root* represents the name of the root directory under which the COMET-AR software system has been installed.

2.12 Model Definition via PATRAN and PST

For most realistic structural models, it is not feasible to construct a model definition procedure manually. Instead, the commercially available PATRAN pre-processing code may be used to generate the model, and the COMET-AR-to-PATRAN conversion processor (PST) is used to translate the PATRAN data to the corresponding model-definition procedure or directly to a COMET-AR database. A description of PATRAN and its usage is beyond the scope of this manual, but Processor PST is described in Section 6.6. Examples of the use of PATRAN and PST to generate a COMET-AR model, as well as on the subsequent solution and post-processing of that model with COMET-AR, may be found in the COMET-AR Tutorial Manual.

2.13 Global Model to Analysis Model Translation Procedure

2.13.1 General Description

This section describes the GM2AM Utility Procedure which calls the GM2AM processor to generate an initial analysis model database from a given 16-node surface-element geometry model and user refinement specifications. The purpose of the GM2AM procedure is to execute the two-phase generation of an initial analysis model from a given 16-node geometry model automatically, by invoking a processor (also called GM2AM) transparently to the user (see Section 6.12 for details on the processor). The GM2AM procedure listing is shown in Box 2.13-1.

Box 2.13-1 Global Model to Analysis Model Translation Procedure

```
*procedure GM2AM ( case              = GENERIC ;--
                   step              = 0       ;--
                   load_set          = 1       ;--
                   constraint_set    = 1       ;--
                   ldi_am            = 2       ;--
                   ldi_gm            = 1           )
.  Execute the INITIALIZE phase
.  ---------------------------------
   run GM2AM
       INITIALIZE
          *add gm2am.add
   stop
.  Open databases files
.  ------------------------
   *open [ldi_am] [case].MODEL.DBC
   *open [ldi_gm] [case].DBG
.  Initialize element and GCP datasets in the analysis database
.  ------------------------------------------------------------------------
   *add init_elt.clp
   *copy [ldi_am] = [ldi_gm], FABRICATIONS
   *copy [ldi_am] = [ldi_gm], MATL.*
.  Execute the REFINE phase
.  ------------------------------
   run GM2AM
       SET LDI_AM              = [ldi_am]
       SET LDI_GM              = [ldi_gm]
       SET STEP                = [step]
       SET LOAD_SET            = [load_set]
       SET CONSTRAINT_SET = [constraint_set]
       REFINE
          *add gm2am.add
   stop
   *close [ldi_am]
   *close [ldi_gm]
*end
```

In addition to supplying the procedure input arguments, the user must also prepare an "add file," called "gm2am.add," which contains user specifications for converting geometric elements into analysis elements to be used as the initial mesh of an adaptive refinement (AR) sequence. See Section 6.12 (*Processor GM2AM*) for details on the preparation of the "gm2am.add" file.

2.13.2 Argument Summary

Procedure GM2AM may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 2.13-1.

Table 2.13-1 Procedure GM2AM Input Arguments

Argument	Default Value	Description
CASE	Generic	Specifies the case name for the geometry and analysis databases
CONSTRAINT_SET	1	Specifies the constraint set number
LDI_AM	2	Specifies the logical device unit for the analysis database file
LDI_GM	1	Specifies the logical device unit for the geometry database file
LOAD_SET	1	Specifies the load set number
STEP	0	Specifies the load- or time-step number

2.13.3 Argument Definitions

In this subsection, the procedure arguments summarized in Table 2.13-2 are defined in more detail. The arguments are listed alphabetically. Refer to Section 6.12 for details on the options.

2.13.3.1 Case Argument

This argument sets the case name prefix for both the geometry and analysis database files.

Argument syntax:

CASE = *case*

where *case* is the file name prefix. The following is the database file naming convention expected by this procedure.

Database	Name Convention
GEOMETRY	case.DBG
ANALYSIS	case.MODEL.DBC

2.13.3.2 CONSTRAINT_SET Argument

This argument defines the constraint set number associated with the element and nodal data in both the geometry and the analysis meshes. This number should appear as the second cycle number in names of all element and nodal datasets.

Argument syntax:

CONSTRAINT_SET = *conset*

where *conset* is the constraint set number (Default value: 1).

2.13.3.3 LDI_AM Argument

This argument sets the logical device index associated with the analysis database file.

Argument syntax:

LDI_AM = *ldi_am*

where *ldi_am* is the logical device index (a positive integer) of the [case].MODEL.DBC file. (Default value: 2).

2.13.3.4 LDI_GM Argument

This argument sets the logical device index associated with the geometry model database file.

Argument syntax:

LDI_GM = *ldi_gm*

where *ldi_gm* is the logical device index (a positive integer) of the [case].DBG file. (Default value: 1).

2.13.3.5 LOAD_SET Argument

This argument defines the load set number associated with the element and nodal data in both the geometry and the analysis meshes. This number should appear as the first cycle number in names of all element and nodal datasets.

Argument syntax:

LOAD_SET = *ldset*

where *ldset* is the load set number (Default value: 1).

2.13.3.6 STEP Argument

This argument defines the solution step number associated with the element and nodal data in both the geometry and the analysis meshes. This number should appear as the first cycle number in names of all element and nodal datasets.

Argument syntax:

STEP = *step*

where *step* is the solution step number (Default value: 0).

2.13.4 Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the GM2Am processor. These dataset requirements are detailed in Section 6.12 on Processor GM2AM.

2.13.5 Current Limitations

GM2AM is a general purpose procedure and the only limitations on its usage are dictated by the limitations of the GM2AM processor, refer to Section 6.12 for details.

2.13.6 Status and Error Messages

GM2AM does not print any status or error messages directly. All messages will be produced by the GM2AM processor; refer to Section 6.12 for specific processor messages.

2.13.7 Examples and Usage Guidelines

```
*call GM2AM (     CASE   =  PCL     )
```

In this example, a complete initial analysis mesh will be generated starting with a 16-node geometry elements database named PCL.DBG and the analysis database will be named PCL.MODEL.DBC.

The user refinement specifications should be provided through the "gm2am.add" file (see Section 6.12, Processor GM2AM, for details). For example, this file may contain the following refinement specifications.

Sample gm2am.add Input File

```
SET ELEMENT_NAME = ES1_EX97
SET P            = 2
SET NEL_X        = 3
SET NEL_Y        = 3
PROCESS_GMELTS   = 0
```

The above "add" file instructs the GM2AM processor to refine every 16-node geometry element present in the geometry database into a 3x3 mesh of 9-node ANS elements in the analysis mesh.

2.13.8 References

None.

Chapter 3 Basic Solution Procedures

3.1 Overview

This chapter describes existing COMET-AR command-language procedures that perform basic finite element solutions (i.e., independent of adaptive mesh refinement). A section is dedicated to each of the currently available procedures listed in Table 3.1-1, including linear static and nonlinear static analysis. Before employing these solution procedures, the user must have first generated a model, as described in the preceding chapter. Then the procedure may be invoked with a simple *CALL directive, after running the COMET-AR macroprocessor (see Chapter 1).

Table 3.1-1 Outline of Chapter 3: Basic Solution Procedures

Section	Procedure	Function
3.1	Overview	Introduction
3.2	L_STATIC_1	Performs linear static analysis
3.3	NL_STATIC_1	Performs nonlinear static analysis

Procedures L_STATIC_1 and NL_STATIC_1 solve the structural equations corresponding to a given finite element mesh. To do this, they employ a number of lower-level (utility) procedures, which in-turn, invoke various processors (described in Part II: Processors).

> All the basic solution procedures described here are also accessible through adaptive solution procedures, such as AR_CONTROL, which perform adaptive mesh refinement in addition to solving the basic equations.

3.2 Procedure L_STATIC_1

3.2.1 General Description

Procedure L_STATIC_1 is a solution procedure for performing linear static analysis. It is automatically invoked by the adaptive refinement AR_CONTROL_1 procedure to perform linear static analysis for a given mesh.

The L_STATIC_1 procedure is merely a simple cover procedure invoking a sequence of utility procedures to perform the linear static analysis task, as shown below in Figure 3.2-1. Each of these utility procedures is discussed in Chapter 5, *Utility Procedures*.

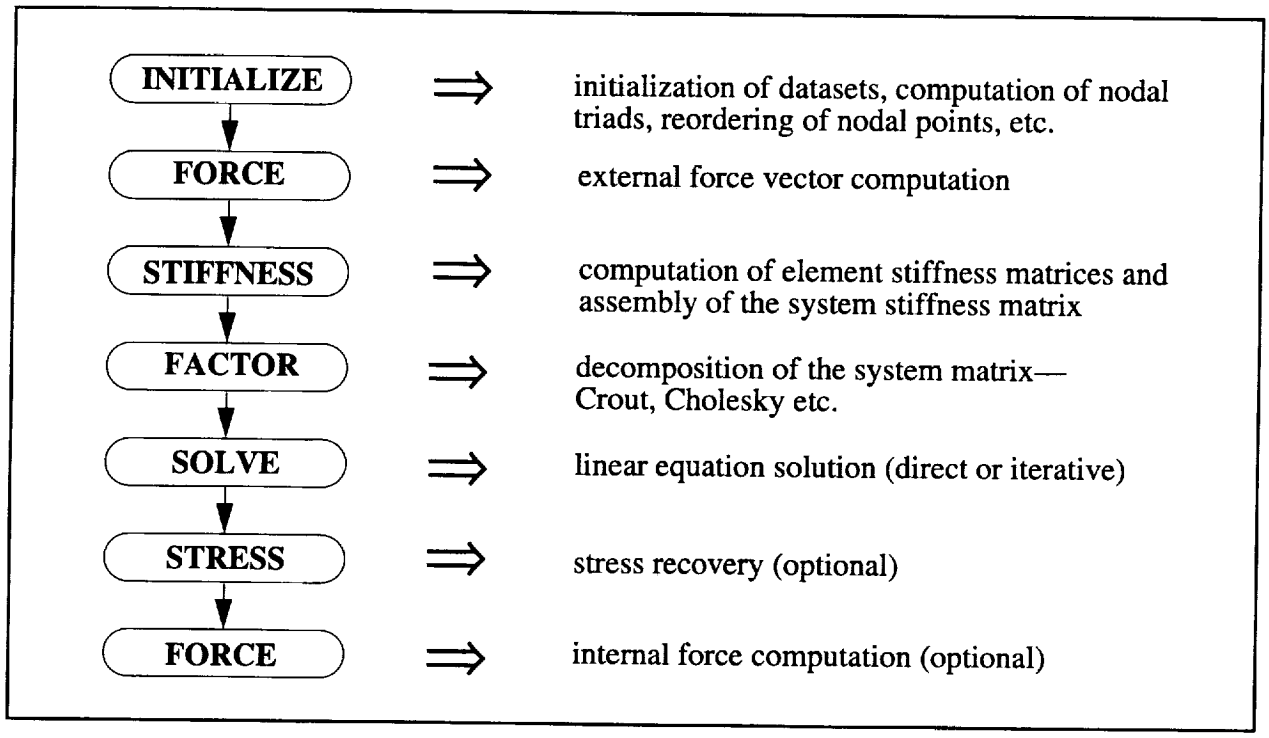


Figure 3.2-1 L_STATIC_1 Algorithm for Linear Finite Element Static Analysis

3.2.2 Argument Summary

Procedure L_STATIC_1 may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 3.2-1.

Table 3.2-1 Procedure L_STATIC_1 Input Arguments

Argument	Default Value	Description
ASM_PROCESSOR	ASM	Matrix/vector assembly processor
AUTO_DOF_SUP	<true>	Automatic DOF suppression switch
AUTO_DRILL	<false>	Automatic drilling stiffness augmentation switch
AUTO_MPC	<false>	
AUTO_TRIAD	<false>	Automatic triad re-alignment for drilling DOFs
CONSTRAINT_SET	1	Constraint set number to be used for suppressing DOFs in the assembled system matrix prior to factorization
FIXED_FRAME	OFF	Fixed-frame option for hierarchical h_s-refinement
INTERNAL	<false>	Compute internal force vector switch
LDI_C	1	Logical unit for main COMET-AR database file (*Case*.DBC)
LDI_E	2	Logical unit for element-matrix file (*Case*.DBE)
LDI_S	3	Logical unit for system-matrix file (*Case*.DBS)
LOAD_SET	1	Load set number to be used as the external force vector
MATRIX_UPDATE	FULL	Matrix update option for hierarchical h_s-refinement
MAX_ITER	100	Maximum iterations for iterative solvers
MESH	0	Mesh number to be analyzed
MTX_BUFFER_SIZE	500000	Matrix buffer size for equation solving
PRINT	<false>	Print solution vector switch
REFINE_TECHNIQUE	ht	Mesh refinement technique (h_t => transition h)
RENO_PROCESSOR	RENO	Node renumbering processor
RENUMBER_OPT	0	Node renumbering option
SKY_PROCESSOR	SKY	Linear equation solver processor name
SOLVER_CONV_TOL	0.000001	Convergence tolerance for iterative solvers
STR_DIRECTION	0	Stress directions for post-processing
STR_LOCATION	INTEG_PTS	
STEP	0	Solution step number
STRESS	<false>	Stress, strain, & strain-energy computation switch

3.2.3 Argument Definitions

In this subsection, the procedure arguments summarized in Table 3.2-1 are defined in more detail. The arguments are listed alphabetically and many of the precise definitions are relegated to subordinate procedures and processors where the actual options are determined. For example, the definition of REFINE_TECHNIQUE depends on which refinement processor the user selects via the REFINE_PROCESSOR argument, so the relevant options can be found in the corresponding refinement processor sections in Part III.

3.2.3.1 ASM_PROCESSOR Argument

Selects the matrix assembly processor to be used for assembling element (stiffness/mass) matrices into corresponding system matrices.

Argument syntax:

ASM_PROCESSOR = *asm_processor*

where *asm_processor* is the name of the matrix assembly processor. Current options include ASM (for h_t and h_c types of mesh refinement) and ASMs (for h_s mesh refinement only). (Default value: ASM.)

3.2.3.2 AUTO_DOF_SUP Argument

Automatic DOF (degree-of-freedom) suppression switch. This capability automatically suppresses extraneous DOFs and is especially useful during adaptive mesh refinement. It is described in more detail in Section 2.10, *Automatic DOF Suppression and Drilling Stabilization.*

Argument syntax:

AUTO_DOF_SUP = *option* [, *angle_tol*]

where

Parameter	Description
option	Automatic DOF suppression option switch: {<true> | <false>}. If <true>, all DOFs (in the computational frame) that are unsupported by element stiffness will be suppressed throughout the adaptive refinement process. (Default value: <true>)
angle_tol	Angle tolerance to use for suppression of shell element drilling DOFs; see Section 2.10 for details. (Default value: depends on element type)

In most cases, it is recommended that the user leave the default setting intact.

3.2.3.3 AUTO_DRILL Argument

Automatic drilling stiffness option. This option causes shell elements to add artificial drilling rotational stiffness to nodal DOFs that would otherwise be unstable computationally. See Section 2.10, *Automatic DOF Suppression and Drilling Stabilization*, and individual element processor sections in Chapter 7, *Element Processors*, for more information.

Argument syntax:

AUTO_DRILL = *option* [, *angle_tol*, *scale_*fac]

where

Parameter	Description
option	Automatic drilling stiffness switch: {<true> | <false>}. If <true>, certain shell element types will add artificial drilling stiffness to nodal DOFs that require stabilization. (Default value: <false>)
angle_tol	Angle tolerance to use for determining whether artificial drilling stiffness is needed at a given node. See Section 2.10 for details. (Default value: depends on element type)
scale_fac	Scale factor determining magnitude of artificial drilling stiffness to be added by selected shell elements. See Chapter 7 for interpretation. (Default value: depends on element type)

AUTO_DRILL is not recommended for nonlinear analysis.

3.2.3.4 AUTO_TRIAD Argument

Automatic computational triad (i.e., DOF direction) re-alignment option. This option is an alternative to AUTO_DRILL that causes re-alignment of the computational triads at all nodes that require drilling DOF stabilization, as long as no boundary conditions have been defined there. The computational axes are re-aligned such that one of them is parallel to the average element surface-normal at the node. Then, extraneous (unstable) drilling rotational DOFs can be subsequently suppressed via the AUTO_DOF_SUP option. (See Section 2.10, *Automatic DOF Suppression and Drilling Stabilization*, for more information.)

Argument syntax:

AUTO_TRIAD = *option* [, *angle_tol*]

where

Parameter	Description
option	Automatic triad re-alignment option switch: {<true> | <false>}. If <true>, computational triads will be re-aligned with the average element normal at all nodes that require drilling DOF stabilization unless boundary conditions are defined there. (Default value: <false>)
angle_tol	Angle tolerance to use for determining whether drilling stabilization is needed at a given node. See Section 2.10 for details. (Default value: depends on element type)

AUTO_TRIAD should only be used in conjunction with AUTO_DOF_SUP and cannot be used in conjunction with user-defined point forces and/or multi-point constraints.

3.2.3.5 CONSTRAINT_SET Argument

This argument defines the constraint set number associated with the element, nodal, and system data. This number should appear as the second cycle number in names of all datasets.

Argument syntax:

CONSTRAINT_SET = *conset*

where:

Parameter	Description
conset	Constraint set number (Default value: 1)

3.2.3.6 FIXED_FRAME Argument

Sets a flag that is relevant only for h_s-refinement.

Argument syntax:

FIXED_FRAME = {<true> | <false>}

Do not change the default setting without the advice of a COMET-AR expert. (Default value: <false>)

3.2.3.7 INTERNAL Argument

This argument sets the internal force computation switch.

Argument syntax:

INTERNAL = *flag*

where *flag* is the switch option. (Default value: <false>. Do not compute internal force.)

3.2.3.8 LDI_C Argument

This argument sets the logical device index associated with the main COMET-AR database file, which must exist before calling L_STATIC_1, and must be named *Case*.DBC.

Argument syntax:

LDI_C = *ldi_c*

where *ldi_c* is the logical device index (a positive integer) of the *Case*.DBC file. (Default value:1)

3.2.3.9 LDI_E Argument

This argument sets the logical device index associated with the element matrix database file, typically named *Case*.DBE.

Argument syntax:

LDI_E = *ldi_e*

where *ldi_e* is the logical device index (a positive integer) of the *Case*.DBE file. If *ldi_e* is not equal to *ldi_c* (see the LDI_C argument) then all element matrices (e.g., stiffness and mass) for the current mesh will be stored on a separate *Case*.DBE file; however, if *ldi_e* = *ldi_c*, then all element matrices will be stored on the *Case*.DBC file, i.e., a separate *Case*.DBE file will not be created. (Default value: 2)

If a separate *Case*.DBE file is created, it will be deleted and re-created with each new adaptive mesh.

3.2.3.10 LDI_S Argument

This argument sets the logical device index associated with the system matrix database file, typically named *Case*.DBS.

Argument syntax:

LDI_S = *ldi_s*

where *ldi_s* is the logical device index (a positive integer) of the *Case*.DBS file. If *ldi_s* is not equal to *ldi_c* (see the LDI_C argument) then all system matrices (e.g., stiffness and mass) for the current mesh will be stored on a separate *Case*.DBS file; however, if *ldi_s* = *ldi_c*, then all system matrices will be stored on the *Case*.DBC file, i.e., a separate *Case*.DBS file will not be created. (Default value: 3)

If a separate *Case*.DBS file is created, it will be deleted and re-created with each new adaptive mesh.

3.2.3.11 MATRIX_UPDATE Argument

This argument sets the matrix-update mode for hierarchical adaptive refinement (relevant only for h_s-refinement).

Argument syntax:

MATRIX_UPDATE = {FULL | PARTIAL}

where FULL implies that the entire stiffness matrix is reformed for each new mesh, and where PARTIAL implies that only the updated-mesh contributions to the stiffness matrix are reformed for each new mesh. (Default value: FULL)

3.2.3.12 MAX_ITER Argument

This argument sets the maximum number of iterations allowed by an iterative linear equation solver (e.g., ITER). Relevant only if SKY_PROCESSOR is set equal to the name of an iterative solver.

Argument syntax:

MAX_ITER = *max_iter*

where *max_iter* is the maximum number of iterations allowed. (Default value: 100)

3.2.3.13 MESH Argument

This argument sets the number of the mesh to analyze.

Argument syntax:

MESH = *mesh*

where *mesh* is the mesh number. (Default value: 0)

3.2.3.14 MTX_BUFFER_SIZE Argument

This argument sets the size of the memory buffer to be used for matrix factorization and solution by certain matrix solution processors.

Argument syntax:

MTX_BUFFER_SIZE = *mtx_buffer_size*

where *mtx_buffer_size* is the size of the buffer in logical variables. (Default value: 500000)

3.2.3.15 PRINT Argument

This argument sets the solution printout switch.

Argument syntax:

PRINT = *flag*

where *flag* is the switch option. (Default value: <false>)

3.2.3.16 REFINE_TECHNIQUE Argument

This argument sets the refinement technique to be employed by the mesh refinement processor (REF*i*) specified via the REFINE_PROCESSOR argument.

Argument syntax:

REFINE_TECHNIQUE = *refine_technique*

where *refine_technique* is the name of the refinement technique. For example, in conjunction with processor REF1, the REFINE_TECHNIQUE argument might be set equal to h_p, h_c, h_s, or p (among others). See documentation under specific REF*i* processors for details. (Default value: h_t)

3.2.3.17 RENO_PROCESSOR Argument

This argument sets the name of the equation (or node) renumbering processor to be used to optimize matrix equation solving (time and/or storage).

Argument syntax:

RENO_PROCESSOR = *renumber_processor*

where *renumber_processor* is the processor name. Current options are summarized below.

renumber_processor	Description
RENO	Node renumbering using a geometric algorithm (Default)
RSEQ	Node renumbering via a variety of order-optimization algorithms

Consult the appropriate sections in Chapter 6, *Pre-Processors*, for more details.

3.2.3.18 RENUMBER Argument

Sets a flag determining whether or not to perform equation renumbering (e.g., bandwidth, skyline, or sparsity optimization) both initially and whenever the mesh is updated by adaptive refinement.

Argument syntax:

RENUMBER = *renumber_flag*

where *renumber_flag* may be set either to <true> or <false>. (Default value: <true>)

3.2.3.19 RENUMBER_OPT

This argument sets the equation renumbering option to use within the renumbering processor selected via the RENO_PROCESSOR argument (assuming RENUMBER = <true>).

Argument syntax:

RENUMBER_OPT = *renumber_option*

where *renumber_option* indicates the renumbering option and depends on the particular renumbering processor chosen. See processors RENO, RSEQ, etc., in Chapter 6, *Pre-Processors*. (Default value: 0)

3.2.3.20 SKY_PROCESSOR Argument

Selects the matrix solution processor to be used for factoring and solving assembled linear equation systems.

Argument syntax:

SKY_PROCESSOR = *sky_processor*

where *sky_processor* is the name of the matrix solution processor. Current options are summarized below.

sky_processor	Description
SKY	Direct solution of skyline matrices by Crout LDU decomposition (Default)
SKYs	Direct and/or iterative solution of skyline matrices in conjunction with h_s-refinement only
ITER	Iterative solution of compact matrices by PCG algorithm
PVSOLV	Direct solution of skyline matrices optimized for vector computers.
VSS	Vectorized sparse solver (very fast and also space-saving).

Consult the appropriate sections in Chapter 12, *Matrix/Vector Processors*, for more details.

3.2.3.21 SOLVER_CONV_TOL Argument

This argument sets the convergence tolerance for the iterative linear equation solver, if one has been selected via the SKY_PROCESSOR argument.

Argument syntax:

SOLVER_CONV_TOL = *solver_conv_tol*

where *solver_conv_tol* is the convergence tolerance. (Default value: 1.e-6)

3.2.3.22 STR_DIRECTION Argument

This argument sets the stress reference frame (x_s,y_s,z_s) for post-processing and/or error estimation purposes.

Argument syntax:

STR_DIRECTION = *str_direction*

where *str_direction* denotes the stress/strain direction. Current options are summarized below:

str_direction	Meaning
ELEMENT (or 0)	Express stress/strain components in the local element (integration point) reference frame (x_s=x_e, y_s=y_e, z_s= z_e). (Default)
GLOBAL {X | Y | Z}	Express stress/strain components in a permutation of the global reference frame, with x_s = x_g, y_g or z_g, if X, Y or Z is selected, respectively. For shell elements, the z_s direction is automatically aligned with the local element normal, z_e, direction.
{1 | 2 | 3}	Same as GLOBAL {X | Y | Z}, respectively.
FAB_DIR	Use the local fabrication axes for the stress frame; i.e., x_s=x_f, y_s=y_f, z_s=y_f. See Section 2.7, *Orientation of Fabrication Reference Frames.*

3.2.3.23 STRESS Argument

Flag determining whether or not element stresses, strains, and strain energy densities are to be computed and stored in the database (Default value: <true>).

Argument syntax:

STRESS = {<true> | <false>}

It is currently necessary to set STRESS=<true> for all analyses involving adaptive mesh refinement.

3.2.4 Database Input/Output Summary

A complete model definition database is required as input for the L_STATIC_1 procedure (see Chapter 2, *Model Definition Procedures*). After the analysis, the solution data will be output to the database for the mesh analyzed; the mesh index will appear as the third index in all dataset names. While most datasets will be stored in the main COMET-AR database *Case*.DBC file, element and system matrices may be stored in the *Case*.DBE and *Case*.DBS files, depending on the user settings for the LDI_E and LDI_S arguments.

3.2.4.1 Input Datasets

Table 3.2-2 contains a list of datasets required (unless otherwise stated) as input by procedure L_STATIC_1. All of these datasets must be resident in the main COMET-AR database file (*Case*.DBC, where *Case* is the specific problem name).

Table 3.2-2 Input Datasets Required by Procedure L_STATIC_1

Dataset	File	Description
CSM.SUMMARY...*mesh*	*Case*.DBC	Model summary for the analyzed mesh
EltName.DEFINITION...*mesh*	*Case*.DBC	Element definition for the analyzed mesh
EltName.FABRICATION...*mesh*	*Case*.DBC	Element fabrication pointers for the analyzed mesh
EltName.GEOMETRY...*mesh*	*Case*.DBC	Element solid-model geometry for the analyzed mesh
EltName.INTERPOLATION...*mesh*	*Case*.DBC	Element interpolation data for the analyzed mesh
EltName.LOAD.*ldcase..mesh*	*Case*.DBC	Element load definition for the analyzed mesh
NODAL.COORDINATE...*mesh*	*Case*.DBC	Nodal coordinates for the analyzed mesh
NODAL.DOF..*conset.mesh*	*Case*.DBC	Nodal DOF Table for the analyzed mesh.
NODAL.TRANSFORMATION...*mesh*	*Case*.DBC	Nodal transformations between global and computational frames for the analyzed mesh
NODAL.SPEC_FORCE.*ldcase..mesh*	*Case*.DBC	Nodal specified forces for the analyzed mesh (optional)
NODAL.SPEC_DISP.*ldcase..mesh*	*Case*.DBC	Nodal specified displacements for the analyzed mesh (optional)

3.2.4.2 Output Datasets

Table 3.2-3 contains a list of datasets that may be created or updated in the database by procedure L_STATIC_1. Most of these datasets will be resident in the main COMET-AR database file (*Case*.DBC), but element and system matrices may be resident in the *Case*.DBE file and *Case*.DBS files, depending on the values of the user-specified arguments LDI_E and LDI_S.

Table 3.2-3 Output Datasets Produced by Procedure L_STATIC_1

Dataset	File	Description
EltName.STRAIN.*ldcase.conset.mesh*	*Case*.DBC	Element strains computed for the analyzed mesh
EltName.STRESS.*ldcase.conset.mesh*	*Case*.DBC	Element stresses computed for the analyzed mesh

Table 3.2-3 Output Datasets Produced by Procedure L_STATIC_1 (Continued)

Dataset	File	Description
EltName.STRAIN_ENERGY.*ldcase.conset.mesh*	*Case*.DBC	Element strain energy densities computed for the analyzed mesh
NODAL.DISPLACEMENT.*ldcase.conset.mesh*	*Case*.DBC	Nodal displacements computed for the analyzed mesh
NODAL.DRILL_FLAG...*mesh*	*Case*.DBC	Nodal suppress drilling DOF flags for the analyzed mesh (optional)
NODAL.EXT_FORCE.*ldcase..mesh*	*Case*.DBC	Nodal external forces for the analyzed mesh
NODAL.NORMAL...*mesh*	*Case*.DBC	Nodal shell normal for the analyzed mesh (optional)
NODAL.ORDER...*mesh*	*Case*.DBC	Nodal re-ordering array, defined by node renumbering processor (optional)
NODAL.DOF..*conset.mesh*	*Case*.DBC	Nodal DOF Table for the analyzed mesh.
SYSTEM.STIFFNESS...*mesh*	*Case*.DBS	System (assembled) stiffness matrix
SYSTEM.VECTOR.*ldcase..mesh*	*Case*.DBS	System (assembled) vector used to store force and displacement vectors during equation solving process.

For details on the contents of any of the above datasets, refer to Chapter 15, *Database Summary.*

3.2.5 Subordinate Procedures and Processors

3.2.5.1 Subordinate Procedures

A list of COMET-AR utility procedures invoked directly by procedure L_STATIC_1 is provided in Table 3.2-4. Documentation may be found in Chapter 5, *Utility Procedures.*

Table 3.2-4 Subordinate Procedures to Procedure L_STATIC_1

Procedure	Type	Function
INITIALIZE	Utility	Performs dataset initialization, node renumbering, etc.
FORCE	Utility	Computes external and internal load vectors
STIFNESS	Utility	Computes element stiffness matrices and assembles the system matrix
FACTOR	Utility	Performs Crout/Cholesky decomposition of the system matrix
SOLVE	Utility	Performs solution of the system linear equations

Table 3.2-4 Subordinate Procedures to Procedure L_STATIC_1 (Continued)

Procedure	Type	Function
STRESS	Utility	Performs stress recovery

3.2.5.2 Relevant Subordinate Processors

Table 3.2-5 lists COMET_AR processors that are invoked directly by procedure L_STATIC_1 and user-specified processors that are invoked indirectly through any of the subordinate procedures listed in Table 3.2-4. (A list of the various non-user-specified processors that are invoked indirectly via subordinate procedures may be obtained by consulting the section on the corresponding procedure.) Documentation on these processors may be found under the chapter on the corresponding processor type.

Table 3.2-5 Relevant Subordinate Processors to Procedure L_STATIC_1

Processor	Type	Function
Assembler	Matrix/Vector	Matrix assembly processor, selected via the ASM_PROCESSOR procedure argument.
Renumbering	Pre-Processor	Equation/node renumbering processor, selected via the RENO_PROCESSOR procedure argument.
Equation Solver	Matrix/Vector	Equation solver, set via the SKY_PROCESSOR argument.

3.2.6 Current Limitations

L_STATIC_1 is a general purpose procedure and the only limitations on its usage, hardware limits, are dictated by the limitations of the procedures and processors being employed. Refer to individual processors and procedures for specific limitations.

3.2.7 Status and Error Messages

L_STATIC_1 does not print any status or error messages directly. All messages will be produces by subordinate procedures and processors invoked during the execution of L_STATIC_1. Refer to individual procedures in Chapter 5, *Utility Procedures*, for further information.

3.2.8 Examples and Usage Guidelines

3.2.8.1 Example 1: Direct Solver

```
*call L_STATIC_1 (      ASM_PROCESSOR       =  ASM          ; --
                        RENUMBER            =  <true>       ; --
                        RENO_PROCESSOR      =  RSEQ         ; --
                        RENUMBER_OPT        =  2            ; --
                        SKY_PROCESSOR       =  SKY          ; --
                        MESH                =  3            ; --
                        STRESS              =  <true>         )
```

In the above example, a linear static analysis is requested for mesh 3. The solution will be obtained using a direct solver (SKY), using the reverse Cuthill-McKee algorithm for profile minimization (RSEQ method 2), and stress recovery will be performed.

3.2.8.2 Example 2: Iterative Solver

```
*call L_STATIC_1 (      ASM_PROCESSOR       =  ASM          ; --
                        RENUMBER            =  <true>       ; --
                        RENO_PROCESSOR      =  RSEQ         ; --
                        RENUMBER_OPT        =  3            ; --
                        SKY_PROCESSOR       =  ITER         ; --
                        MAX_ITER            =  2000         ; --
                        SOLVER_CONV_TOL     =  1.0e-7       ; --
                        MESH                =  2            ; --
                        STRESS              =  <true>         )
```

In the above example, a linear static analysis is requested for mesh 2. The solution will be obtained using an iterative solver (ITER) with maximum number of iterations=2000 and solver convergence tolerance set to 1.0e-7. Node renumbering will be performed using the reverse Gibbs-Poole-Stockmeyer algorithm for bandwidth minimization (RSEQ method 3) and stress recovery will be performed.

3.2.9 References

[1] Stanley, G., Levit, I., Hurlbut, B., and Stehlin, B. *Adaptive Refinement (AR) Strategies for Shell Structures, Part 1: Preliminary Research*, Preliminary NASA Contract Report, 1991.

[2] Stehlin, B., *The COMET-AR User's Tutorial*, NASA Preliminary Contract Report, February, 1993.

3.3 Procedure NL_STATIC_1

3.3.1 General Description

Procedure NL_STATIC_1 is a solution procedure for performing nonlinear static analysis, including both geometrical and material nonlinearity using an arclength-controlled version of a modified Newton-Raphson incremental/iterative nonlinear solution algorithm (see [1] and [2]). This procedure enables the automatic traversal of limit points and quasi-bifurcation points, which are commonly experienced in the postbuckling/failure analysis of structures. The user must provide an initial load factor, a maximum/minimum load factor, and a set of strategy parameters (most of which have reasonable default values), and the procedure will attempt to obtain an automatic solution to the problem within the number of load steps and other limits specified by the user.

The equations solved by procedure NL_STATIC_1 are the nonlinear static equilibrium equations:

$$\mathbf{r}(\mathbf{d}, \lambda) = \mathbf{f}^{ext}(\lambda) - \mathbf{f}^{int}(\mathbf{d}) = \mathbf{0}$$

where $\mathbf{d}$ is the system displacement vector, λ is an external load factor, $\mathbf{f}^{ext}$ is the scaled external force vector, $\mathbf{f}^{int}$ is the internal force vector, and $\mathbf{r}$ is the residual force vector. The above equations are also subjected to the following scalar arclength constraint equation:

$$c(\mathbf{d}) = \|\Delta\mathbf{d}\|^2 - \|\Delta l\|^2 = 0$$

where Δ denotes an increment between two successive load steps (e.g., λ_n and λ_{n+1}), and l is an arclength parameter, approximating the distance along the load-displacement curve (see Figure 3.3-1).

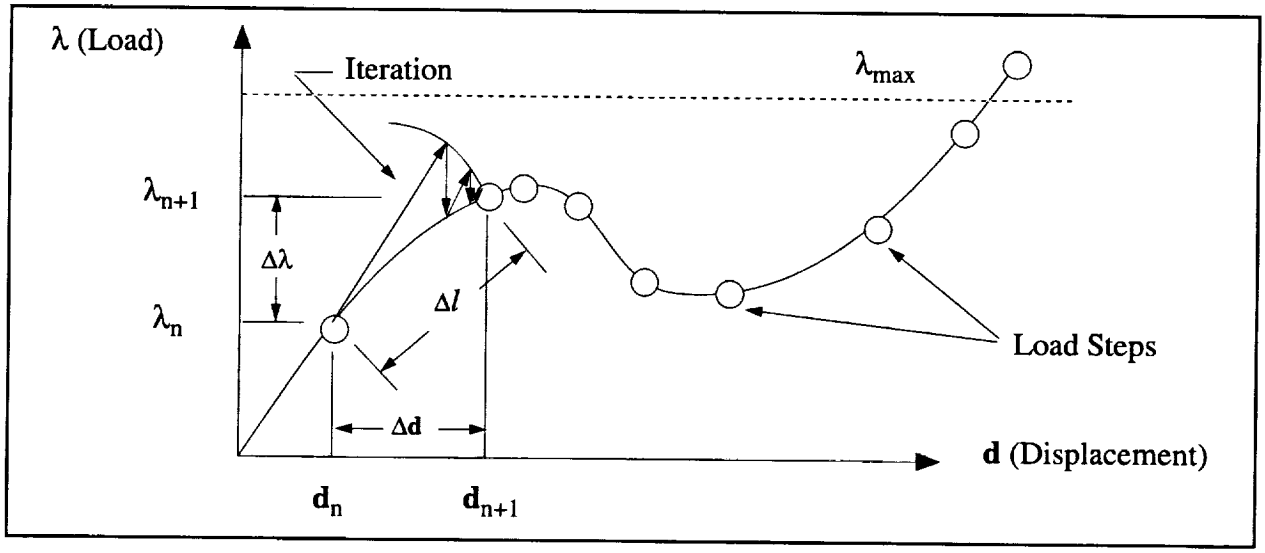


Figure 3.3-1 Typical Load-Displacement Curve Computed by NL_STATIC_1

The solution of these two equations involves their linearization about the current configuration (which may or may not be in equilibrium) with an iteration loop to obtain convergence in the neighborhood of that configuration. An outer step loop advances the solution along the load-displacement curve (analogous to load-step incrementation in load-controlled versus arclength-controlled solution algorithms). The essential features of the solution algorithm are illustrated in Figure 3.3-2. For more details, consult references [3]–[5].

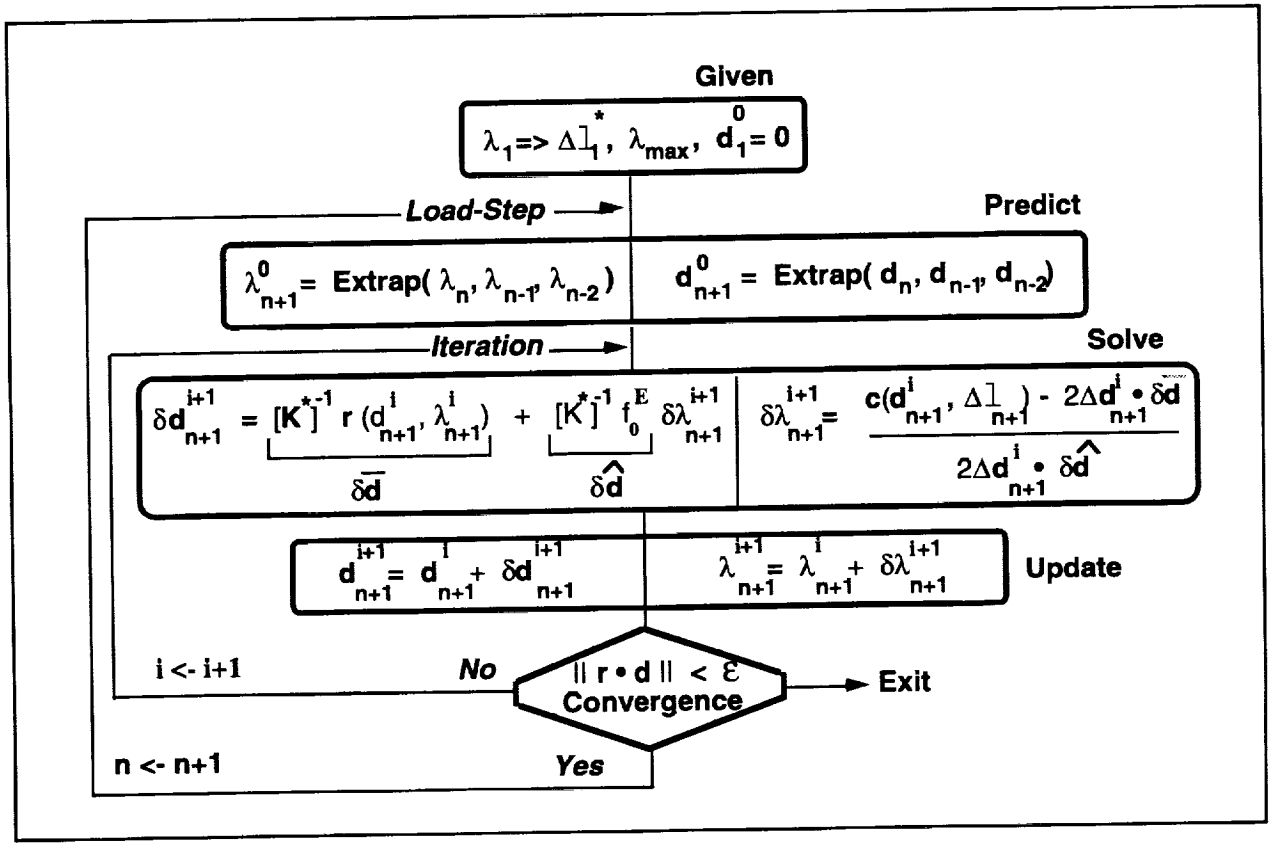


Figure 3.3-2 Overview of Procedure NL_STATIC_1 Solution Algorithm

In Figure 3.3-2, **K** denotes the "effective" tangent stiffness (which in the modified Newton-Raphson algorithm is updated only once every one or more load steps), ε denotes a user-specified nonlinear error tolerance, the subscript "n" denotes the load step number, and the superscript "i" denotes the iteration number within a given load step. The user provides the starting and stopping conditions, and the rest is automatic: at each new load step, both the new displacement vector ($\mathbf{d}_{n+1}$) and the new load factor (λ_{n+1}) are predicted via quadratic extrapolation along the solution path. Then, within the iteration loop, two iterative-change displacement vectors are computed: $\delta\bar{\mathbf{d}}$ and $\delta\hat{\mathbf{d}}$; the first is based on the residual force vector as right-hand-side, the second is based on the external force vector as right-hand side. The arclength constraint equation then enables the calculation of the corresponding iterative change in the load factor, $\delta\lambda$, which in turn provides the necessary ingredients to compute the combined iterative displacement-change vector, $\delta\mathbf{d}$. Finally, both the displacement vector and the load factor are updated by their respective iterative changes,

and convergence is checked based on the inner product of the residual force vector (**r**) and the iterative displacement change (δ**d**) through an energy error norm.

3.3.2 Argument Summary

Procedure NL_STATIC_1 may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 3.3-1. These procedure arguments are partitioned into mandatory and optional categories. It is assumed that all necessary database files have been opened via the *OPEN directive before calling NL_STATIC_1.

Table 3.3-1 Procedure NL_STATIC_1 Input Arguments

Argument	Default Value	Description
MANDATORY Arguments		
BEG_LOAD		Starting load factor
MAX_LOAD		Upper bound on load factor
MIN_LOAD		Lower bound on load factor
OPTIONAL Arguments		
AUTO_DOF_SUP	<true>, 0	Automatic DOF suppression option
AUTO_DRILL	<false>, 0, 0	Automatic drilling stiffness augmentation option
AUTO_MPC	<false>	Automatic "drilling" multipoint constraint switch
AUTO_TRIAD	<false>, 0	Automatic triad re-alignment for drilling DOFs
ASM_PROCESSOR	ASM	Matrix/vector assembly processor name
BEG_STEP	1	Starting step number (>0)
CONSTRAINT_SET	1	Number of boundary condition set to employ
CONV_CRITERIA	CHKCONV_E	
COROTATION	<true>	Corotational option for large rotations
DES_ITERS	4	Desired number of iterations per load step
DSN_R	RESPONSE.HISTORY	Name of selected-results dataset
EXTRAPOLATE	<true>	Quadratic predicted solution extrapolation flag
FAC_STEPS	1	Number of steps between stiffness refactoring
INITIALIZE	<true>	Optional initialization flag for solution restarts
INTERPOLATE	<false>	Mesh interpolation flag for adaptive refinement
LAST_REF_STEP	1	Last step refined (AMR)
LDI_C	1	Logical device index for main COMET-AR file
LDI_E	1	Logical device index for element matrix file
LDI_R	1	Logical device index for selected results file

Table 3.3-1 Procedure NL_STATIC_1 Input Arguments

Argument	Default Value	Description
LDI_S	1	Logical device index for system matrix file
LINE_SEARCH	1	Initial line-search parameter
LOAD_SET	1	Number of load set to employ in analysis
LOAD_STIFF	<false>	Include load stiffness
MAX_CUTS	3	Maximum number of automatic step cuts
MAX_ITERS	9	Maximum number of iterations per load step
MAX_STEPS	1	Maximum number of load steps to compute
MESH	0	Mesh number to analyze (from linear AR)
N_SELECT	0	Number of nodal DOFs for selected archival
NEWTON	<false>	Toggle for TRUE Newton iteration
NL_GEOM	2	Geometric nonlinearity option
NL_MATL	0	Material nonlinearity option
NL_TOL	1.e-3	Relative error tolerance for nonlinear convergence
PATH_SCALE	0.0	Arclength scale factor to use for restarts (0=automatic)
POST	0	
REFINE	<false>	Refinement flag
RENO_PROCESSOR	RSEQ	Node renumbering processor
RENUMBER_OPT	3	Node renumbering option
SEL_DOFS		List of DOF numbers for selected archival
SEL_NODES		List of node numbers for selected archival
SKY_PROCESSOR	SKY	Linear equation solver processor name
SOLVER_MAX_ITER	1000	Maximum iterations for iterative solvers
SOLVER_CONV_TOL	0.000001	Convergence tolerance for iterative solvers
STR_DIRECTION	0	Stress/strain reference frame for post-processing
STR_LOCATION	INTEG_PTS	
STRESS	<true>	Stress/strain database archival step frequency flag
ARCHIVE_STEP	10	Archival step frequency for nonlinear material data

3.3.3 Argument Definitions

In this subsection, the procedure arguments summarized in Table 3.3-1 are defined in more detail. The arguments are listed alphabetically, and some of the precise definitions are relegated to subordinate procedures and processors (covered elsewhere in this manual) where the actual options are determined.

3.3.3.1 ARCHIVE_STEP Argument

This argument sets the load step frequency for database archival of nonlinear material historical data.

Argument syntax:

ARCHIVE_STEP = *step_frequency*

where *step_frequency* is a non-negative integer indicating that nonlinear material historical data should be archived every "step_frequency"th load step. The value of *step_frequency* determines at which load steps the solution can be re-started, i.e., historical data must be archived at a given step in order for the solution to be continued in a re-start run from that step. A value of 0 implies that no archival will be performed for the current solution interval. Relevant only for materially non-linear analysis. (Default value: 10)

3.3.3.2 ASM_PROCESSOR Argument

This argument selects the matrix assembly processor to be used for assembling element stiffness and mass matrices into corresponding system matrices.

Argument syntax:

ASM_PROCESSOR = *asm_processor*

where *asm_processor* is the name of the matrix assembly processor. The current option is limited to processor ASM. (Default value: ASM)

3.3.3.3 AUTO_DOF_SUP Argument

This argument defines the automatic DOF (degree-of-freedom) suppression option. This capability automatically suppresses extraneous DOFs not supported by element stiffness. It is described in more detail in Section 2.10, *Automatic DOF Suppression and Drilling Stabilization.*

Argument syntax:

AUTO_DOF_SUP = *option* [, *angle_tol*]

where:

Parameter	Description
option	Automatic DOF suppression option switch: {<true> | <false>}. If <true>, all DOFs (in the computational frame) unsupported by element stiffness will be suppressed throughout the adaptive refinement process. (Default value: <true>)

Parameter	Description
angle_tol	Angle tolerance to use for suppression of shell element drilling DOFs; see Section 2.10 for details. (Default value: depends on element type)

In most cases, it is best to leave the default setting intact.

3.3.3.4 AUTO_DRILL Argument

This argument defines the automatic drilling stiffness option. This option causes shell elements to add artificial drilling rotational stiffness to nodal DOFs that would otherwise be unstable computationally. See Section 2.10, *Automatic DOF Suppression and Drilling Stabilization*, and individual element processor sections in Chapter 7, *Element Processors,* for more information.

Argument syntax:

AUTO_DRILL = *option* [, *angle_tol* , *scale_*fac]

where:

Parameter	Description
option	Automatic drilling stiffness switch: { <true> | <false> }. If <true>, certain shell element types will add artificial drilling stiffness to nodal DOFs that require stabilization. (Default value: <false>)
angle_tol	Angle tolerance to use for determining whether artificial drilling stiffness is needed at a given node. See Section 2.10 for details. (Default value: depends on element type)
scale_fac	Scale factor determining magnitude of artificial drilling stiffness to be added by selected shell elements. See individual Element Processor section in Chapter 7 for interpretation. (Default value: depends on element type)

AUTO_DRILL is not recommended for nonlinear analysis.

3.3.3.5 AUTO_MPC Argument

This argument sets the automatic multi-point constraint (MPC) option for suppressing extraneous drilling DOFs, defined as rotations about the normal to a plate or shell element. Unless the element has intrinsic stiffness associated with such rotations, these DOFs may lead to a singular stiffness matrix. Turning the AUTO_MPC option on causes special constraints to be generated at nodes where insufficient drilling rotational stiffness is present, to suppress the rotation about the appropriate ("drilling") axis. This axis is generally not aligned with any of the computational axes, and so the constraint will typically involve a linear combination of the rotational DOFs cor-

responding to the computational axes. See Section 2.10, *Automatic DOF Suppression and Drilling Stabilization*, for more information on this option and related options such as AUTO_DOF_SUP, AUTO_DRILL, and AUTO_TRIAD.

Argument syntax:

AUTO_MPC = *option* [, *angle_tol*]

where:

Parameter	Description
option	Automatic multi-point constraint switch for drilling stabilization: { <true> | <false> }. If <true>, multi-dof constraints will be generated at nodes where drilling stabilization is needed. (Default value: <false>)
angle_tol	Angle tolerance to use for determining whether drilling stabilization is needed at a given node. See Section 2.10 for details. (Default value: depends on element type)

3.3.3.6 AUTO_TRIAD Argument

This argument defines the automatic computational triad (i.e., DOF direction) re-alignment option. This option, an alternative to AUTO_DRILL, causes re-alignment of the computational triads at all nodes that require drilling DOF stabilization, as long as no boundary conditions have been defined there. The computational axes are re-aligned such that one of them is parallel to the average element surface-normal at the node. Then, extraneous (unstable) drilling rotational DOFs can be subsequently suppressed via the AUTO_DOF_SUP option. (See Section 2.10, *Automatic DOF Suppression and Drilling Stabilization*, for more information.)

Argument syntax:

AUTO_TRIAD = *option* [, *angle_tol*]

where:

Parameter	Description
option	Automatic triad re-alignment option switch: { <true> | <false> }. If <true>, computational triads will be re-aligned with average element normal at all nodes that require drilling DOF stabilization, unless boundary conditions are defined there. (Default value: <false>)
angle_tol	Angle tolerance to use for determining whether drilling stabilization is needed at a given node. See Section 2.10 for details. (Default value: depends on element type)

AUTO_TRIAD should only be used in conjunction with AUTO_DOF_SUP, and should not be used in conjunction with user-defined point forces and/or multi-point constraints.

3.3.3.7 BEG_LOAD Argument

This argument sets the starting load factor, λ_1, for the nonlinear analysis.

Argument syntax:

BEG_LOAD = λ_1

For applied force loading, this factor is multiplied by the reference (i.e., base) applied force vector, $\mathbf{f}_0^{ext}$, i.e.,

$$\mathbf{f}_1^{ext} = \lambda_1 \mathbf{f}_0^{ext}$$

where $\mathbf{f}_0^{ext}$ is a combination of the user's specified nodal (concentrated) forces in dataset NODAL.SPEC_FORCE.*ldset..mesh* and specified element (distributed) forces in dataset *Elt-Nam*.LOAD.*ldset..mesh*. For applied displacement loading, the starting load factor is applied to the reference (i.e., base) user-specified displacement vector, $\mathbf{d}_0^{ext}$ stored in dataset NODAL.SPEC_DISP.*ldset..mesh* which is then used to compute the initial internal force vector, $\mathbf{f}_1^{int}$. In either case, the starting load factor is used to compute the starting arclength increment, Δl_1, which is then modified adaptively (see DES_ITERS argument) while the load factor becomes a solution variable throughout the rest of the analysis. (Default value: None)

This argument is irrelevant for re-start runs (i.e., BEG_STEP>1) in which case the PATH_SCALE argument is used (indirectly) to determine how big of a load step to take.

3.3.3.8 BEG_STEP Argument

This argument sets the number of the first load step to be computed in a given nonlinear analysis interval.

Argument syntax:

BEG_STEP = *beg_step*

where *beg_step* is the beginning (or starting) step number. Initially, *beg_step* should be set equal to 1. For analysis re-start runs, *beg_step* should be set equal to the next step to compute (or re-

compute). For example, if *n* steps had been computed (and saved in the database) during the initial run, the user would set *beg_step* equal to *n*+1 for a re-start run that continues where the first run left off. It is not necessary for *beg_step* to be larger than any previously computed step. That is, the user may wish to recompute a sequence of steps by setting *beg_step* equal to the number of the first step to be re-computed. Procedure NL_STATIC_1 will then automatically use the solution data for those steps immediately preceding step *beg_step* (i.e., *beg_step–1*, *beg_step–2*, and *beg_step–3*) to smoothly effect the restart, and over-write the solution data for each re-computed load step (i.e., *beg_step*, *beg_step+1*, ... up to the highest step originally computed). (Default value: None)

3.3.3.9 CONSTRAINT_SET Argument

This argument specifies the number of the boundary condition constraint set (defined by the user during Model Definition) to be employed for the current nonlinear analysis.

Argument syntax:

CONSTRAINT_SET = *conset*

where *conset* is the constraint set number. (Default value: 1)

3.3.3.10 COROTATION Argument

This argument selects the element corotational update option to be employed by the generic element processor for geometrically nonlinear analysis (i.e., large displacements and rotations, small to moderate strains).

Argument syntax:

COROTATION = *corotation*

where *corotation* is the option number, for which valid entries are given below.

corotation	Description
0	Element corotational updates will not be used to account for large rotation effects; any such effects must therefore be handled by the element's own nonlinear strain-displacement relations, activated by setting the argument NL_GEOM= 2.
1	Basic element corotational updates will be used to account for large rotation effects. The accuracy of this approach can be enhanced if nonlinear element strain-displacement relations are used (by setting argument NL_GEOM=2) but linear element strain-displacement relations (NL_GEOM=1) are acceptable if the mesh is sufficiently fine. (Default)
2	Higher-order element corotational option. This is essentially the same as option 1, except some additional terms are added to the element stiffness matrix which can increase the rate of nonlinear convergence, but only in conjunction with true Newton iteration (see NEWTON argument).

Corotation is a built-in feature of the Generic Element Processor (see Section 7.2, *Generic Element Processor*), which subtracts the bulk rigid-body motion from each element (via the element corotational reference frame described in Section 2.2, *Reference Frames and Coordinate Systems*), leaving deformational displacements and rotations that are relatively small (and become smaller as the mesh is refined), regardless of how large the bulk motions (i.e., total displacements and rotations) are. This allows elements that are based on only moderate rotation theory (e.g., most beam and shell elements), or even infinitesimal rotation theory, to be applied to problems involving arbitrarily large rotations but small strains. For theoretical details on the corotational method implemented in COMET-AR, refer to the Generic Element Processor Manual [4]; for a description of how corotation interacts with procedure NL_STATIC_1, refer to the section on NL_STATIC_1 in reference [3].

3.3.3.11 DES_ITERS Argument

This argument sets the desired number of iterations for nonlinear convergence at each load step, which affects how the step size is adaptively updated during the run.

Argument syntax:

DES_ITERS = *des_iters*

where *des_iters* is the desired number of iterations. The step-size update algorithm is as follows. If the actual number of iterations required to obtain convergence at step n is within the limit set by the MAX_ITERS argument, then the new arclength increment for step $n+1$ is defined in terms of the arclength increment at step n via the linear relationship:

$$\Delta l_{n+1} = \frac{desired_iters}{actual_iters} \times \Delta l_n$$

If the actual number of iterations is larger than the desired number, the new step size will be proportionally smaller, and conversely if the actual number of iterations is smaller than the desired number, the new step size will be proportionally larger then the old step size. Only if the actual number of iterations is identical to the desired number does the step size remain constant. (Default value: 4)

3.3.3.12 DSN_R Argument

This argument specifies the name of the dataset within the results database file (also see argument LDI_R) where selected results and nonlinear solution parameter values are to be stored.

Argument syntax:

DSN_R = *dsn_r_name*

where *dsn_r_name* is the name of the dataset. (Default value: RESPONSE.HISTORY)

3.3.3.13 EXTRAPOLATE Argument

This argument sets a flag determining whether or not to use quadratic extrapolation along the solution path to predict the load factor and displacement vector at the beginning of each load step.

Argument syntax:

EXTRAPOLATE = { <true> | <false> }

where <true> implies that quadratic extrapolation will be used.The use of quadratic extrapolation is recommended since it has been found to be a very effective strategy for accelerating traversal of the load-displacement curve. Far fewer load steps are usually required with extrapolation than without except at abrupt slope discontinuities in the curve, where a quadratic polynomial is too smooth to be of much help. (Default value: <true>)

The EXTRAPOLATE=<false> option has not been fully tested, and hence is not recommended.

3.3.3.14 FAC_STEPS Argument

This argument sets the number of load steps between stiffness matrix updates (i.e., re-forming and re-factoring).

Argument syntax:

FAC_STEPS = *fac_steps*

where *fac_steps* is a positive integer indicating that re-factoring of a new stiffness matrix will be performed every *fac_steps* loads step. For modified Newton iteration, the stiffness update will be performed only at the beginning of such steps; for true Newton iterations, the stiffness update will be performed at each iteration of the step. (The NEWTON argument may be used to select modified versus true Newton iteration.) Best results are often obtained with *fac_steps* set to 1. (Default value: 1)

3.3.3.15 INITIALIZE Argument

This argument sets a flag determining whether or not to initialize element parameters, constitutive parameters, and equation numbers when performing a solution re-start.

Argument syntax:

INITIALIZE = { <true> I <false> }

where <true> implies that initialization will be performed at the beginning of the current solution interval, and <false> implies that it will not be performed. For the very first solution interval (i.e., starting at step 1), the initialization flag should be set to <true>. For subsequent re-starts, it should be set to <false> unless adaptive refinement has been performed, an option which has not yet been fully tested. For now, use the default value of <true> initially, and change it to <false> for all subsequent re-start runs. (Default value: <true>)

3.3.3.16 INTERPOLATE Argument

This argument sets an interpolation flag option that can be used in conjunction with adaptive mesh refinement. It is typically invoked automatically when NL_STATIC_1 is called by an adaptive solution control procedure such as AR_CONTROL_1.

Argument syntax:

INTERPOLATE = { <false> I <true> }

where <true> means that the predicted displacement solution for step BEG_STEP will be obtained by spatially interpolating from the solution for the previous mesh. (Default: <false>)

3.3.3.17 LDI_C Argument

This argument sets the logical device index associated with the main COMET-AR database file, typically named *Case*.DBC.

Argument syntax:

LDI_C = *ldi_c*

where *ldi_c* is the logical device index (a positive integer) of the .DBC file. (Default value: 1)

The .DBC file must be opened by the user via an "*OPEN *ldi_c*" directive before invoking procedure NL_STATIC_1.

3.3.3.18 LDI_E Argument

This argument sets the logical device index associated with the element matrix database file, typically named *Case*.DBE.

Argument syntax:

LDI_E = *ldi_e*

where *ldi_e* is the logical device index (a positive integer) of the .DBE file. (Default value: 1)

To create a .DBE file separate from the .DBC (main COMET-AR database) file, it must be opened/created via an "*OPEN *ldi_e*" directive before invoking procedure NL_STATIC_1; otherwise, if *ldi_e* = *ldi_c*, all element matrices will be stored in the .DBC file.

3.3.3.19 LDI_R Argument

This argument sets the logical device index associated with the selected results database file, typically named *Case*.DBR.

Argument syntax:

LDI_R = *ldi_r*

where *ldi_r* is the logical device index (a positive integer) of the .DBR file. This file will be used to store all user-selected displacement results, as well as key solution parameters, in a dataset whose name is specified by the DSN_R argument. (Default value: 1)

To create a .DBR file separate from the .DBC file, it must be opened/created via an "*OPEN *ldi_r*" directive before invoking NL_STATIC_1; otherwise, if *ldi_r* = *ldi_c*, the selected results will be stored in the .DBC file.

3.3.3.20 LDI_S Argument

This argument sets the logical device index associated with the system matrix database file, typically named *Case*.DBS.

Argument syntax:

LDI_S = *ldi_s*

where *ldi_s* is the logical device index (a positive integer) of the .DBS file. (Default value: 1)

To create a .DBE file separate from the .DBC (main COMET-AR database) file, it must be opened/created via an "*OPEN *ldi_s*" directive before invoking procedure NL_STATIC_1; otherwise, if *ldi_s* = *ldi_c*, all system matrices will be stored in the .DBC file.

3.3.3.21 LOAD_SET Argument

This argument specifies the number of the external load set (defined by the user during Model Definition) to be employed for the current nonlinear analysis.

Argument syntax:

LOAD_SET = *ldset*

where *ldset* is the load set number. (Default value: 1)

3.3.3.22 MAX_CUTS Argument

This argument sets the maximum number of step cuts allowed during the current nonlinear run.

Argument syntax:

MAX_CUTS = *max_cuts*

where *max_cuts* is the maximum number of cuts allowed. A step cut refers to a halving of the arclength increment, Δl, used to advance the solution from one step to the next. Step cuts are performed only if the maximum number of iterations (specified via the MAX_ITERS argument) is exceeded without converging at a given step. (Default value: 3)

> Whenever the step is cut, a new displacement predictor is extrapolated, and a corresponding new stiffness matrix is formed and factored unless the user has turned off the solution extrapolation switch (via the EXTRAPOLATE argument). The user may manually introduce arbitrary step size reductions (or increases) by stopping the analysis and re-starting with a modified value of the PATH_SCALE argument.

3.3.3.23 MAX_ITERS Argument

This argument sets the maximum number of iterations allowed for nonlinear convergence at a given load step.

Argument syntax:

MAX_ITERS = *max_iters*

where *max_iters* is the maximum number of iterations allowed. If *max_iters* iterations have been performed at a given step, and nonlinear convergence (to an equilibrium state) has not yet been obtained, procedure NL_STATIC_1 will attempt to cut the step size as many times as allowed via

the argument MAX_CUTS. If the limits set by both MAX_CUTS and MAX_ITERS have been reached, then the run will be terminated and the user will have to try a different strategy (see Section 3.3.9, *Usage Guidelines*). (Default value: 9)

3.3.3.24 MAX_LOAD Argument

This argument sets the maximum load factor, λ_{max}, for the nonlinear analysis.

Argument syntax:

MAX_LOAD = λ_{max}

The value λ_{max} establishes an upper limit on the load level, and provides a convenient way of terminating the arclength-controlled solution algorithm. Since the load factor is actually a solution variable (i.e., an unknown) in procedure NL_STATIC_1, there is no way of knowing a priori how many load steps will be required to attain λ_{max}. The analysis is terminated when either λ_{max}, λ_{min}, *max_steps* or *max_cuts* is exceeded as set by the MAX_LOAD, MIN_LOAD, MAX_STEPS, and MAX_CUTS arguments, respectively. (Default value: None)

Procedure NL_STATIC_1 may overshoot the maximum load factor somewhat, as it does not fix the last load increment to force convergence to the user-specified maximum.

3.3.3.25 MAX_STEPS Argument

This argument sets the maximum number of load steps to compute during the current nonlinear analysis run with procedure NL_STATIC_1.

Argument syntax:

MAX_STEPS = *max_steps*

where *max_steps* is the maximum number of steps to compute in the current run, not to be confused with the number of the highest load step in the analysis. This provides an implicit limit on analysis run time. Since the load factor is actually a solution unknown (controlled by the arclength parameter, Δl) there is no way of knowing a priori how many load steps will be required to attain the user's designated maximum or minimum load factor (specified via the MAX_LOAD and MIN_LOAD arguments). The nonlinear run is terminated whenever MAX_STEPS, MAX_LOAD, MIN_LOAD, or MAX_CUTS is exceeded. (Default value: None)

3.3.3.26 MESH Argument

This argument sets the number of the mesh to be analyzed throughout the nonlinear analysis.

Argument syntax:

MESH = *mesh*

where *mesh* is the mesh number. Unless linear adaptive mesh refinement has been performed earlier (i.e., via solution procedure AR_CONTROL_1) the mesh number will always be 0. The current capabilities for adaptive mesh refinement during the nonlinear analysis are experimental and not recommended for general use. (Default value: 0)

3.3.3.27 MIN_LOAD Argument

This argument sets the minimum load factor, λ_{min}, for the nonlinear analysis.

Argument syntax:

MIN_LOAD = λ_{min}

The value λ_{min} establishes a lower limit on the load level which should be less than the starting load factor, λ_1, specified by the BEG_LOAD argument. This provides a convenient way of terminating the arclength-controlled solution algorithm. Since the load factor is actually a solution variable (i.e., an unknown) in procedure NL_STATIC_1, there is no way of knowing a priori how many load steps will be required to attain λ_{min}. The analysis is terminated when either λ_{max}, λ_{min}, *max_steps* or *max_cuts* is exceeded as set by the MAX_LOAD, MIN_LOAD, MAX_STEPS, and MAX_CUTS arguments. (Default value: None)

> Procedure NL_STATIC_1 may undershoot the maximum load factor somewhat, as it does not fix the last load increment to force convergence to the user-specified minimum.

3.3.3.28 NEWTON Argument

This argument determines the type of Newton-Raphson iteration algorithm to use: modified or true.

Argument syntax:

NEWTON = { MODIFIED | TRUE }

If NEWTON=MODIFIED, stiffness matrix updates (re-forming and re-factoring) will be performed only at the beginning of every *fac_steps* load steps, where *fac_steps* is set via the

FAC_STEPS argument. If NEWTON=TRUE, stiffness matrix updates will be performed at each iteration of every *fac_steps* load steps. (Default value: MODIFIED)

> Modified Newton iteration is typically more effective than true Newton iteration except at critical junctures of the solution trajectory, where dramatic changes are taking place rapidly, e.g., mode switching, bifurcation-like behavior, contact, and abrupt material damage such as progressive crack or delamination growth.

3.3.3.29 NL_GEOM Argument

This argument selects the geometrical nonlinearity option to be used in the current analysis run.

Argument syntax:

NL_GEOM = *nl_geom*

where *nl_geom* is the option number, for which valid entries are given below.

nl_geom	Description
0	The problem is treated as geometrically linear, i.e., infinitesimally small displacements, rotations and strains.
1	The problem is geometrically nonlinear, but only linear strain-displacement relations will be used at the element level; it is assumed that large displacements and rotations will be handled via the corotational option (see COROTATION argument).
2	The problem is geometrically linear, and nonlinear strain-displacement relations will be used at the element level whether or not the corotational option is selected by the user (see COROTATION argument). (Default)

Option 2 is generally more accurate than option 1, but requires the particular element type selected to have the capability for nonlinear strain-displacement relations (refer to the descriptions of specific element processors in Chapter 7, *Element Processors*). For more information on the corotational capability, refer to the COROTATION argument as well as references [3] and [4].

3.3.3.30 NL_MATL Argument

This argument selects the material nonlinearity option to be used in the current analysis run.

Argument syntax:

NL_MATL = *nl_matl*

where *nl_matl* is the option number, for which valid entries are given below.

nl_matl	Description
0	Material nonlinearity will not be considered. (Default)
1	Material nonlinearity will be considered, provided the material types selected by the user during model definition are based on a nonlinear constitutive model.

3.3.3.31 NL_TOL Argument

This argument sets the value of the error tolerance used to establish convergence of the nonlinear equilibrium iteration process at each load step.

Argument syntax:

NL_TOL = *nl_tol*

where *nl_tol* is the relative error tolerance in the energy error norm. The iteration loop at a given load step is terminated whenever the following condition is met:

$$\varepsilon \leq nl_tol$$

where ε is an error norm that may be selected via the NL_CONV_CRITERIA argument, e.g.,

$$\varepsilon = \sqrt{\frac{\mathbf{r}^i \bullet \delta\mathbf{d}^i}{\mathbf{r}^1 \bullet \delta\mathbf{d}^1}}$$

is the relative energy error norm, where $\mathbf{r}$ is the residual force vector, $\delta\mathbf{d}$ is the iterative displacement change vector, and i is the iteration counter at a given step. (Default value: 1.e-3)

3.3.3.32 N_SELECT Argument

This argument specifies the number of user-selected displacement components to be saved in the results database (see arguments LDI_R and DSN_R for specification of the results database file number and dataset name).

Argument syntax:

N_SELECT = *n_select*

where *n_select* is the number of displacement components to save. The actual node and DOF numbers identifying these displacement components are specified via the SEL_NODES and SEL_DOFS arguments. (Default value: 0)

3.3.3.33 NL_CONV_CRITERIA

This argument selects the command-language procedure to be used to assess nonlinear convergence at each iteration of a nonlinear analysis.

Argument syntax:

NL_CONV_CRITERIA = *procedure_name*

where the options are:

procedure_name	Description
CHKCONV_E	Uses strain-energy norm as convergence measure, with incremental quantity as a reference value in the denominator for relative error. (Default)
CHKCONV_SE	Same as CHKCONV_E except employs total strain energy (square root) as reference value in the denominator to obtain relative error.
CHKCONV_D	Uses Euclidean norm of displacement vector change as error measure, with norm of total displacement in denominator.

3.3.3.34 PATH_SCALE Argument

This argument sets a scale factor to be applied to the current arclength increment, Δl, for the first step in a re-start run. It is thus a manual way to adjust the solution step size.

Argument syntax:

PATH_SCALE = *path_scale*

where *path_scale* is a non-negative floating point number. If *path_scale* is set to 1., the step-size from the previous step (i.e., *beg_step–1*, where *beg_step* is set by the BEG_STEP argument) will be used to compute the first new step (*beg_step*), i.e.,

$$\Delta l_{n+1} = path_scale \times \Delta l_n$$

where n = *beg_step–1*. This may lead to a different step size than would have been obtained had the analysis continued without a re-start since the step-size would have been adjusted based on the ratio of desired-to-actual iterations (see the DES_ITERS argument). The main function of this

argument is for the user to override the procedure's step-size adjustment algorithm, in cases where the user has a better idea based on experience. (Default value: 0 => use automatic step-size adjustment algorithm)

3.3.3.35 SEL_DOFS Argument

This argument specifies a list of DOF numbers designating user-selected displacement components to be saved in the results database (see arguments LDI_R and DSN_R for specification of the results database file number and dataset name). The number of displacement components to be that are saved is set with the N_SELECT argument. The node numbers are set via the SEL_NODES argument.

Argument syntax:

SEL_DOFS = *DOF_1, DOF_2, . . ., DOF_N_SELECT*

where *DOF_1, DOF_2, ...*, represent nodal DOF numbers (e.g., ranging between 1 and 6 for standard 6 DOF per node problems) and *N_SELECT* represents the number of components selected via the N_SELECT argument. (Default value: 0)

For each of the N_SELECT displacement components selected there is a node and DOF number pair, set via the SEL_NODES and SEL_DOFS arguments. For example, to save all 6 DOFs at node 10 in the selected results dataset, the user would set:

N_SELECT=6, SEL_NODES=6@10, and SEL_DOFS=1:6.

3.3.3.36 SEL_NODES Argument

This argument specifies a list of node numbers for user-selected displacement components to be saved in the results database (see arguments LDI_R and DSN_R for specification of the results database file number and dataset name). The number of displacement components that are to be saved is set with the N_SELECT argument.

Argument syntax:

SEL_NODES = *node_1, node_2, . . ., node_N_SELECT*

where *node_1, node_2, ...*, represent global node numbers, and *N_SELECT* represents the number of components selected with the N_SELECT argument. (Default value: 0)

Only those DOFs (i.e., components) selected via the SEL_DOFS argument will be stored for these nodes.

3.3.3.37 SKY_PROCESSOR Argument

Selects the matrix solution processor to be used for factoring and solving assembled linear equation systems.

Argument syntax:

SKY_PROCESSOR = *sky_processor*

where *sky_processor* is the name of the matrix solution processor. Current options are summarized below.

sky_processor	Description
SKY	Direct solution of skyline matrices by Gauss elimination (Default)
SKYs	Direct and/or iterative solution of skyline matrices in conjunction with h_s refinement only
ITER	Iterative solution of compact matrices by PCG algorithm
PVSOLV	Direct solution of skyline matrices optimized for vector computers.
VSS	Vectorized sparse solver (very fast and also space-saving)

Consult Chapter 12, *Matrix/Vector Processors*, for details on individual solution processors.

3.3.3.38 STR_DIRECTION Argument

Sets the stress/strain reference frame (x_s,y_s,z_s) for post-processing and/or error estimation purposes.

Argument format:

STR_DIRECTION = *str_direction*

where *str_direction* denotes the stress/strain direction. Current options are summarized below.

str_direction	Meaning
ELEMENT (or 0)	Express stress/strain components in the local element (integration point) reference frame (x_s=x_l, y_s=y_l, z_s= z_l). (Default)
GLOBAL { X | Y | Z }	Express stress/strain components in a permutation of the global reference frame, with x_s = x_g, y_g or z_g, if X, Y or Z is selected, respectively. For shell elements, the z_s direction is automatically aligned with the local element normal, z_l, direction.
{ 1 | 2 | 3 }	Same as GLOBAL {X | Y | Z } respectively.

str_direction	Meaning
FAB_DIR	Use local fabrication axes for the stress frame, i.e., $x_s=x_f$, $y_s=y_f$, $z_s=y_f$. See Section 2.7, *Orientation of Fabrication Reference Frames.*

3.3.3.39 STRESS Argument

Flag determining whether or not element stresses, strains, and strain energy densities are to be computed and stored in the database. (Default value: <true>)

Argument format:

STRESS = { <true> | <false> }

It is currently necessary to set STRESS=<true> for all analyses involving adaptive mesh refinement.

3.3.4 Database Input/Output Summary

A complete model definition database is required as input for the first run with procedure NL_STATIC_1 (see Chapter 2, *Model Definition Procedures*). After the analysis, solution result data, e.g., displacements, stresses, internal forces, etc., will have been output to the database for each load step computed. In addition, intermediate solution data, such as incremental displacement vectors, residual force vectors, element stiffness matrices and system stiffness matrices for the current (i.e., most recently computed) step will be stored in the database. Most of the datasets will be stored in the main COMET-AR database (.DBC file, associated with argument LDI_C), while the element matrices may be stored in the .DBE file, and the system matrices may be stored in the .DBS file, depending on the values set by arguments LDI_E and LDI_S, respectively.

3.3.4.1 Input Datasets

Table 3.3-2 contains a list of datasets required (unless otherwise stated) as input by procedure NL_STATIC_1. All of these datasets must be resident in the main COMET-AR database, *Case*.DBC, which is assumed to be open and attached to the logical device index specified by the

LDI_C argument. The variables *mesh*, *ldset*, and *conset* represent the mesh index, load set, and constraint set number, respectively.

Table 3.3-2 Input Datasets Required by Procedure NL_STATIC_1

Dataset	File	Description
CSM.SUMMARY...*mesh*	*Case*.DBC	Model summary
EltName.DEFINITION...*mesh*	*Case*.DBC	Element definition
EltName.FABRICATION...*mesh*	*Case*.DBC	Element fabrication pointers
EltName.GEOMETRY...*mesh*	*Case*.DBC	Element solid-model geometry
EltName.INTERPOLATION...*mesh*	*Case*.DBC	Element interpolation data
EltName.LOAD.*ldset..mesh*	*Case*.DBC	Element load definition
NODAL.COORDINATE...*mesh*	*Case*.DBC	Nodal coordinates
NODAL.DOF..*conset.mesh*	*Case*.DBC	Nodal DOF boundary conditions
NODAL.TRANSFORMATION...*mesh*	*Case*.DBC	Nodal transformation matrices (global ->computational)
NODAL.SPEC_FORCE.*ldset..mesh*	*Case*.DBC	Nodal specified forces
NODAL.SPEC_DISP.*ldcase..mesh*	*Case*.DBC	Nodal specified displacements

3.3.4.2 Output Datasets

Table 3.3-3 contains a list of datasets that are created/stored in the database by procedure NL_STATIC_1. Most of these datasets will be resident in the central COMET-AR database file *Case*.DBC associated with argument LDI_C, but element and system matrices may be resident in the *Case*.DBE and *Case*.DBS files, depending on the values of the user-specified arguments LDI_E and LDI_S. Selected displacement components and solution strategy parameters will be stored in a *RESPONSE.HISTORY* dataset (specified via the DSN_R argument) that will either be resident on the .DBC file or on a separate .DBR file, depending on the values associated with arguments LDI_R and LDI_C.

Table 3.3-3 Output Datasets Updated/Created by Procedure NL_STATIC_1

Dataset(s)	File	Description
CSM.SUMMARY...*mesh*	*Case*.DBC	Model summary with updated load step counter
EltNam.STIFFNESS...*mesh*	*Case*.DBE	Element matrices for current step
EltNam.STRAIN.*step..mesh*	*Case*.DBC	Element strains at each step
EltNam.STRESS.*step..mesh*	*Case*.DBC	Element stresses at each step
EltNam.STRAIN_ENERGY.*step..mesh*	*Case*.DBC	Element strain energy densities at each step
NODAL.DISPLACEMENT.*step..mesh*	*Case*.DBC	Nodal displacements at each step

Table 3.3-3 Output Datasets Updated/Created by Procedure NL_STATIC_1

Dataset(s)	File	Description
NODAL.ORDER..*conset.mesh*	*Case*.DBC	Nodal re-ordering array (optional)
NODAL.DOF..*conset.mesh*	*Case*.DBC	Nodal DOF dataset updated with equation numbers
NODAL.EXT_FORCE...*mesh*	*Case*.DBC	Nodal external force vector at current step
NODAL.INC_DISP...*mesh*	*Case*.DBC	Nodal incremental disp. vectors, $\Delta\mathbf{d}$, at current step
NODAL.INC_DISP_BAS...*mesh*	*Case*.DBC	Nodal incremental disp. vectors, $\delta\bar{\mathbf{d}}$, at current step
NODAL.INC_DISP_ITR...*mesh*	*Case*.DBC	Nodal incremental disp. vectors, $\delta\mathbf{d}$, at current step
NODAL.INC_DISP_TAN...*mesh*	*Case*.DBC	Nodal incremental disp. vectors, $\delta\hat{\mathbf{d}}$, at current step
NODAL.INT_FORCE.*step..mesh*	*Case*.DBC	Nodal internal force vectors at each step
NODAL.RES_FORCE...*mesh*	*Case*.DBC	Nodal residual force vector at current step
NODAL.ROTATION.*step..mesh*	*Case*.DBC	Nodal rotation pseudo-vectors at each step
NODAL.TAN_FORCE...*mesh*	*Case*.DBC	Nodal tangent force vector at current step
RESPONSE.HISTORY	*Case*.DBR	History of selected displacements and solution parameters
STRUCTURE.STIFFNESS...*mesh*	*Case*.DBS	Latest assembled structural stiffness matrix

3.3.5 Subordinate Procedures and Processors

Procedure NL_STATIC_1 employs a number of utility procedures to perform nonlinear static analysis, and these procedures, in turn, employ a number of processors to perform most of the calculations. An overview of the subordinate utility procedures is given in Figure 3.3-3. The following sections list and summarize the functions of both subordinate procedures and processors

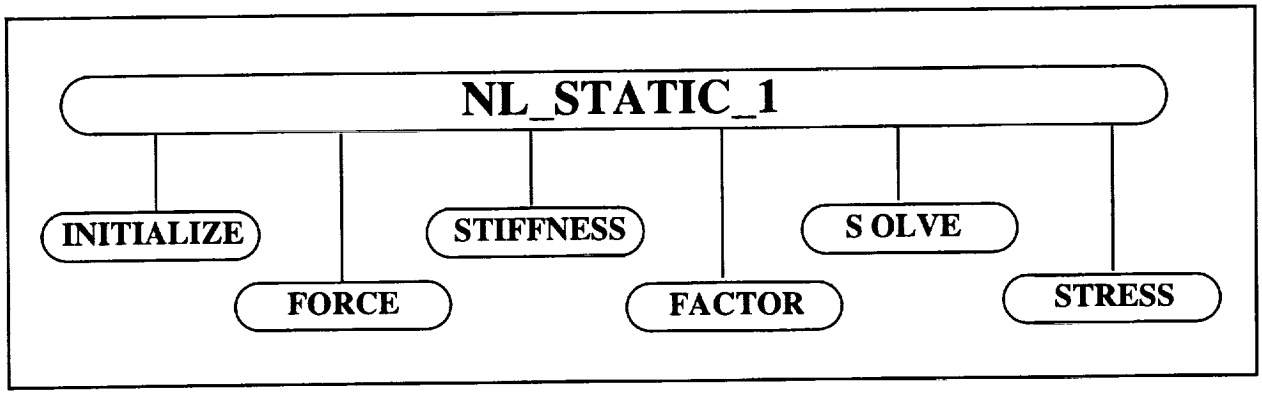


Figure 3.3-3 Organization of Procedure NL_STATIC_1

3.3.5.1 Subordinate Procedures

Utility procedures invoked directly by procedure NL_STATIC_1 is provided in Table 3.3-4.

Table 3.3-4 Subordinate Procedures to Procedure NL_STATIC_1

Procedure	Type	Function
FACTOR	Utility	Factors assembled structural stiffness matrix
FORCE	Utility	Forms and assembles structural force vectors
INITIALIZE	Utility	Initializes element data and assigns equation numbers
SOLVE	Utility	Solves linear equation systems using factored stiffness
STIFFNESS	Utility	Forms and assembles structural stiffness matrix
STRESS	Utility	Computes element stresses, strains, and strain energies

Documentation on these procedures may be found in Chapter 5, *Utility Procedures*.

3.3.5.2 Subordinate Processors

A list of COMET_AR processors that are invoked by procedure NL_STATIC_1 and its utility procedures is given in Table 3.3-5.

Table 3.3-5 Subordinate Processors to Procedure NL_STATIC_1

Processor	Type	Calling Procedures	Function
ASM_Processor	Matrix/ Vector	STIFFNESS SOLVE	Assembles element stiffness matrices into structural stiffness matrix. Assembles force vector due to specified displacement components.
RENO	Pre-Processor	INITIALIZE	Renumbers nodes so as to achieve "optimal" equation numbers for linear solver.
SKY_Processor	Matrix/ Vector	FACTOR SOLVE	Linear equation solver, set via procedure argument: SKY_PROCESSOR.
ES*i*	Element	INITIALIZE FORCE STIFFNESS STRESS	Relevant element processors (invoked indirectly, via utility procedure ES) perform all element related functions.
COP	Pre-Processor	INITIALIZE SOLVE	Defines nodal boundary conditions and multi-point constraints.
TRIAD	Special-Purpose	INITIALIZE	Re-aligns computational triads at nodes if AUTO_TRIAD procedure argument is on.

Table 3.3-5 Subordinate Processors to Procedure NL_STATIC_1

Processor	Type	Calling Procedures	Function
VEC	Matrix/ Vector	NL_STATIC_1	Vector utility processor, used for all vector algebra operations, including dot products, "saxpy's," and even nodal pseudo-vector updates for large rotations.

Documentation on these processors may be found under the chapter on the corresponding processor type.

3.3.6 Current Limitations

A summary of current limitations is given in Table 3.3-6.

Table 3.3-6 Current Limitations of Procedure NL_STATIC_1

Limitation		Description	Work-Around
1	Single Load System	Only one load system is currently allowed. This means that all load contributions are scaled by the same load factor.	None.
2	Not all Element Types	Not all element types are equipped for nonlinear analysis, especially geometrically nonlinear analysis. Check the documentation on the specific element type selected and make sure that the element has both geometric stiffness matrix and internal force vector capabilities implemented (under the Status and Limitations subsection of the appropriate Element Processor section).	Select only elements that have the necessary capabilities.
3	Limit on Specified Rotations	Specified non-zero rotational DOFs are currently valid only in cases where the specified rotation components remain smaller than about 10 degrees.	Make an effort to employ translational DOFs only for specified displacement loading.
4	Cannot Fix Load Increments	Load increments are computed automatically by the algorithm, and cannot be influenced by the user except indirectly, by changing the arclength step size (via the PATH_SCALE and DES_ITERS arguments).	None.
5	Not Foolproof	There is no such thing as a fully automatic nonlinear solution algorithm. The unexpected is to be expected, and the user may have to try all kinds of different strategies to complete the analysis.	Be an actively involved user; question all results, and experiment with solution strategies to gain experience. Consult Usage Guidelines, references [3] and [5], and more experienced users.

3.3.7 Status and Error Messages

A summary of important status and error messages potentially printed by Procedure NL_STATIC_1 is given in Table 3.3-7.

Table 3.3-7 Status and Error Messages for Procedure NL_STATIC_1

Status/Error Message		Potential Cause(s)	Suggested User Response
1	Non-Convergence at Step n. Revise Strategy	The maximum number of nonlinear iterations (MAX_ITERS) has been exhausted, as well as the maximum number of step cuts (MAX_CUTS), and convergence still has not been obtained at step n.	Try re-starting the analysis from several steps back, and decrease the arclength increment at that point (using the PATH_SCALE). Or, just increase MAX_ITERS and MAX_CUTS
2	Divergence at Step n. Revise Strategy.	This message has implications like the previous message, but occurs when the error grows instead of decreases during two successive nonlinear iterations. The difference between divergence and non-convergence is that divergence cannot be cured by increasing MAX_ITERS. It generally means that the step size is too big.	Try re-starting from an earlier step, and reduce the size of the arclength increment via argument PATH_SCALE, and/or the error tolerance via TOL_E.
3	Convergence Difficulties; Repeating Step n with Reduced Path_Increment	Convergence has not been obtained for step n within the maximum number of iterations allowed (see MAX_ITERS argument). The procedure is cutting the step size (i.e., arclength increment) in half and will try again.	Relax. It is normal for the step to have to be cut several times beyond the initial estimate, especially during the more nonlinear stages of the load-displacement history.
4	Convergence at Step n	The solution at load step n has converged to an equilibrium state within the user's error tolerance. The procedure is ready to advance to the next load step.	All is probably well, for now. Remember that nonlinear problems can have multiple solutions, and there is no guarantee that you won't have to recompute step n later, especially if you find that you are on an unstable equilibrium path (see Usage Guidelines).

3.3.8 Usage Examples

3.3.8.1 Example 1: Starting a Nonlinear Analysis

```
*call NL_STATIC_1 (      BEG_STEP      = 1       ; --
                         MAX_STEPS     = 20      ; --
                         BEG_LOAD      = 0.1     ; --
                         MAX_LOAD      = 1.0       )
```

In the above example, a new nonlinear analysis is started with only the minimum necessary information provided by the user: the starting step number (1), the starting load factor (.1), the maximum number of steps (20), and the maximum load factor (1.). All other parameters will take on

their default values (see Argument Definitions). It is implicitly assumed in this example that the model has been defined and stored on a .DBC database file connected to logical device index 1 (LDI_C); that all data will be stored on this database file (including element and system matrices); that modified Newton iterations will be allowed, up to 9 per step; that 3 step cuts will be allowed per step; etc. At the end of the run, the solution may get as far as step 20 or the maximum load, whichever comes first. On the other hand, it may have gotten stuck prematurely at some earlier step or load factor, in which case a solution re-start will be needed (see next example).

3.3.8.2 Example 2: Re-Starting (or Continuing) a Nonlinear Analysis

```
*call NL_STATIC_1 (     BEG_STEP      = 11      ; --
                        MAX_STEPS     = 100     ; --
                        MAX_LOAD      = 1.0     ; --
                        PATH_SCALE    = 0.2     ; --
                        DES_ITERS     = 3         )
```

The above example is a sequel to Example 1, and assumes that in the first run, convergence difficulties were encountered, say at step 15, after trying the default of 3 step cuts and 9 iterations. In the re-start run, the user forces a smaller step size to be taken starting from step 11. The PATH_SCALE=.2 setting indicates that the arclength increment used for step 10 is to be divided by 5 before re-computing step 10; and the DES_ITERS=3 setting will help to keep the step sizes smaller than before, by requiring convergence to occur in 3 iterations per step, rather than the default which is 4 (see the DES_ITERS argument description for an explanation of how this argument is used to control the step size). In the above re-start, the original steps 11 through 15 (which were computed in Example 1) will be over-written with new versions of these steps which may correspond to totally different load levels. Various other changes in solution strategy parameters may be effective for analysis re-starts; it is case-dependent.

For detailed examples of nonlinear analysis performed with procedure NL_STATIC_1, consult the COMET-AR Tutorial [6] and the COMET Applications Manual [7]. While the latter reference is based on an earlier generation of the code (COMET-BL), the operation of procedure NL_STATIC_1 is still very much the same in COMET-AR.

3.3.9 Usage Guidelines

Guidelines for performing nonlinear analysis with COMET-AR procedure NL_STATIC_1 can be found in the *CSM Nonlinear Analysis Tutorial* [5].

3.3.10 References

[1] Riks, E., "An Incremental Approach to the Solution of Snapping and Buckling Problems," *International Journal of Numerical Methods in Engineering*, Vol. 15, pp. 524–551, 1979.

[2] Crisfield, M. A., "A Fast Incremental/Iterative Solution Procedure that Handles Snap-through," *Computers and Structures*, Vol. 13, pp. 55–62, 1983.

[3] Stewart, C. B. (ed.), *The Computational Structural Mechanics (CSM) Testbed Procedures Manual*, preliminary NASA Technical Memorandum, May, 1990.

[4] Stanley, G. M. and Nour-Omid, S., *The Computational Structural Mechanics (CSM) Testbed Generic Element Processor Manual*, NASA CR-181728, March, 1990.

[5] Stanley, G. M., *CSM Nonlinear Analysis Tutorial*, presentation given at NASA Langley Research Center (hand-outs available), December, 1992.

[6] Stehlin, B., *The COMET-AR User's Tutorial*, preliminary NASA Contract Report, February, 1993.

[7] Hurlbut, B. J., Stanley, G. M. and Kang, D. S., *The Computational Structural Mechanics (CSM) Testbed Applications Manual*, preliminary NASA Contract Report, May, 1989.

Chapter 4 Adaptive Solution Procedures

4.1 Overview

This chapter describes existing COMET-AR command-language procedures for performing adaptive finite element solutions, i.e., structural analysis with adaptive mesh refinement. A section is dedicated to each one of these control procedures, as listed in Table 4.1-1.

Table 4.1-1 Outline of Chapter 4: Adaptive Solution Procedures

Section	Procedure	Function
4.1	Overview	Introduction
4.2	AR_CONTROL	Controls adaptive linear/nonlinear static analysis

Currently there is only one adaptive solution procedure, AR_CONTROL, and it is restricted to linear static analysis, i.e., it invokes the basic solution procedure L_STATIC_1 described in the previous chapter. In general, adaptive solution procedures invoke basic solution procedures, as illustrated in Figure 4.1-1.

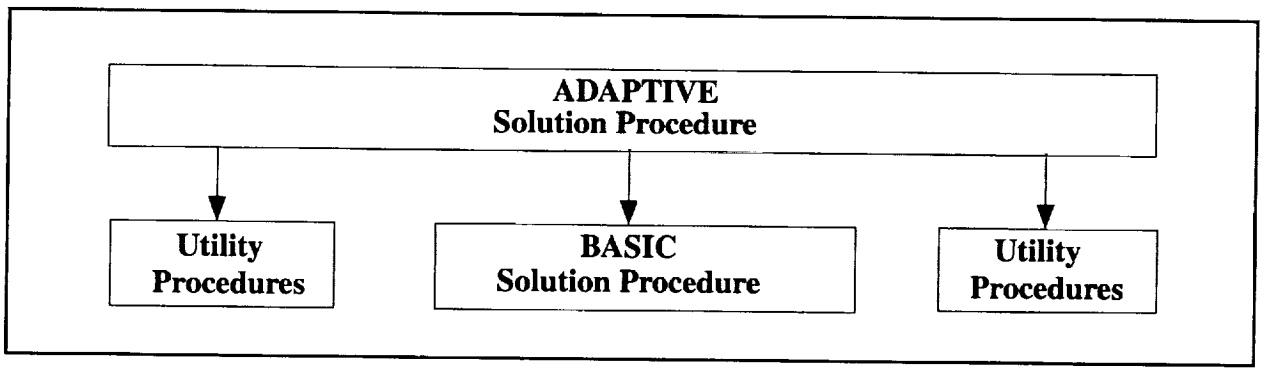


Figure 4.1-1 Relationship Between Adaptive and Basic Solution Procedures

To employ an adaptive solution procedure, the user must have first generated a model (as described in Chapter 2, *Model Definition Procedures*). The adaptive solution procedure may then be invoked via a simple *CALL directive while running the COMET-AR macro-processor.

4.2 Procedure AR_CONTROL

4.2.1 General Description

Procedure AR_CONTROL is a solution procedure for performing linear and nonlinear static analysis with (or without) adaptive mesh refinement. It automatically invokes the appropriate basic solution procedures, L_STATIC_1 or NL_STATIC_1 (see Chapter 3, *Basic Solution Procedures*) to perform linear or nonlinear static analysis for a given mesh, followed (optionally) by utility procedures EST_ERR_* and REF_MESH_* to estimate element errors and perform adaptive mesh refinement. This process can be carried out iteratively, as shown in Figure 4.2-1, by proper choice of input parameters until spatial convergence to a user-specified tolerance has been obtained.

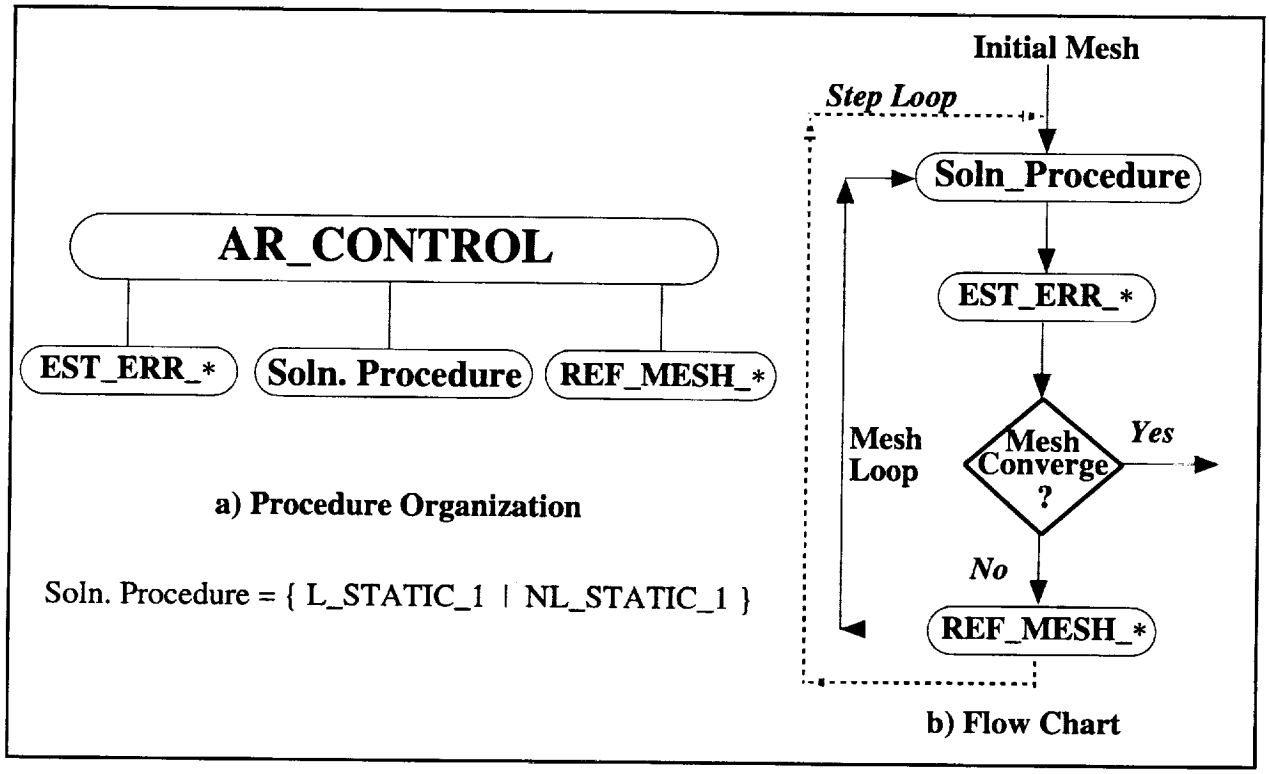


Figure 4.2-1 AR_CONTROL Procedure for Linear/Nonlinear Adaptive Mesh Analysis

The Step Loop in Figure 4.2-1b is relevant only for nonlinear analysis. Most of the actual work is performed by the various COMET-AR processors described in Part III, and reference to these processors will be made where appropriate.

AR_CONTROL may be employed as a common user interface to perform linear or nonlinear analysis without interacting directly with L_STATIC_1 or NL_STATIC_1. Guidelines/examples for performing adaptive mesh refinement with linear static analysis are given in the COMET-AR Tutorial [2]. Capabilities for performing adaptive mesh refinement with nonlinear static analysis are very preliminary, and should be invoked only by experienced researchers.

4.2.2 Argument Summary

Procedure AR_CONTROL may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 4.2-1. These procedure arguments are partitioned into the following groups for the user's convenience: i) Model Control; ii) Basic Solution Control; iii) Nonlinear Solution Control; iv) Error Estimation Control; and v) Mesh Refinement Control arguments.

Table 4.2-1 Procedure AR_CONTROL Input Arguments

Argument	Default Value	Description
MODEL CONTROL Arguments		
CASE	AR_TEST	Case name (first name of database files)
LDI_C	1	Logical unit for main database file (*Case*.DBC)
LDI_E	2	Logical unit for element-matrix file (*Case*.DBE)
LDI_S	3	Logical unit for system-matrix file (*Case*.DBS)
LDI_R	4	Logical unit for selected results file (*Case*.DBR)
LDI_GM	7	
LOAD_SET	1	Load set number to be analyzed
CONSTRAINT_SET	1	Constraint set number to be analyzed.
BASIC SOLUTION CONTROL Arguments		
SOLN_PROCEDURE	<false>	Name of solution procedure
AUTO_DOF_SUP	<true>	Automatic DOF suppression switch
AUTO_DRILL	<false>	Automatic drilling stiffness augmentation switch
AUTO_MPC	<false>	Automatic "drilling" multipoint constraint switch
AUTO_TRIAD	<false>	Automatic triad re-alignment for drilling DOFs
RENO_PROCESSOR	RENO	Node renumbering processor
RENUMBER_OPT	0	Node renumbering option
ASM_PROCESSOR	ASM	Matrix/vector assembly processor
FIXED_FRAME	OFF	Fixed frame option for hierarchical h_s-refinement
MATRIX_UPDATE	FULL	Matrix update option for hierarchical h_s-refinement
MTX_BUFFER_SIZE	512000	Matrix buffer size for equation solving
SKY_PROCESSOR	SKY	Linear equation solver processor name

Table 4.2-1 Procedure AR_CONTROL Input Arguments (Continued)

Argument	Default Value	Description
SOLVER_MAX_ITER	100000	Maximum iterations for iterative solvers
SOLVER_CONV_TOL	0.000001	Convergence tolerance for iterative solvers
STRESS	<false>	Stress, strain and strain energy archival switch
STR_DIRECTION	0 (element local frame)	Stress directions (frames) for post-processing
STR_LOCATION	INTEG_PTS	Stress locations for post-processing
INTERNAL_FORCE	<false>	Internal force archival switch
N_SELECT	0	Number of selected displacement components for archival in LDI_R file
SEL_NODES	0	List of selected nodes for archival in LDI_R file
SEL_DOFS	0	List of selected DOFs for archival in LDI_R file
POST	<true>	Special post-processing procedure switch
ERROR ESTIMATION CONTROL Arguments		
ERROR_PROCESSOR	<false>	Name of error estimation processor to invoke
ERROR_TECHNIQUE	S/BARLOW	Error estimation technique (S => Smoothing)
ERROR_MEASURE	*strain_energy*	Solution quantity upon which errors are based
ERROR_FREQUENCY	1	Number of steps between error estimations
SAMPLE_LOCATIONS	INTEG_PTS	
SMOOTH_PROCESSOR	SMZ	Name of smoothing processor for error estimates
SMOOTH_LOCATIONS	INTEG_PTS	Smoothing evaluation locations
SMOOTH_OPTIONS	0.	Special option list for smoothing processor
NUM_GROUP	0	Number of element groups for error estimation
ELEMENT_GROUPS	0	List of element groups for error estimation
MESH REFINEMENT CONTROL Arguments		
BEG_MESH	0	Starting mesh for AR iteration loop
MAX_MESHES	1	Stopping mesh for AR iteration loop
OLD_MESH	0	Mesh to restart from
CONVERGE_TOL	.05	Global error tolerance (relative error)
REFINE_PROCESSOR	<false>	Name of mesh refinement processor
REFINE_TECHNIQUE	ht	Mesh refinement technique (h_t => transition h)
REFINE_INDICATOR	MAX_RATIO	Type of refinement indicator
REFINE_DIRS	1, 2	Refinement directions (1,2—implies 2D)
NUM_REFINE_TOLS	1	Number of error tolerances guiding refinement
REFINE_TOLS	.90	List of local (element) error tolerances for refinement
REFINE_LEVELS	1	List of refinement levels corresponding to REFINE_TOLS

Table 4.2-1 Procedure AR_CONTROL Input Arguments (Continued)

Argument	Default Value	Description
NUM_UNREFINE_TOLS	0	No. of error tolerances guiding refinement
UNREFINE_TOLS	.00	List of local (element) error tolerances for refinement
UNREFINE_LEVELS	0	List of refinement levels corresponding to REFINE_TOLS
MAX_ASPECT_RATIO	0, 0	Distortion control parameters for h_t-refinement
MAX_H_LEVEL	10	Maximum levels of h-refinement for any element
MAX_P_LEVEL	5	Maximum levels of p-refinement globally
BEG_STEP_REF	1	Nonlinear load step at which to begin mesh refinement
NUM_STEP_REF	1	Number of nonlinear load steps between mesh refinement loops
MAX_MESH_STEP	0	Maximum number of mesh updates per step
LAST_REF_STEP	1	Last step at which mesh refinement was performed
NONLINEAR SOLUTION CONTROL Arguments		
BEG_STEP	1	Starting load step for nonlinear solution interval
MAX_STEPS	30	Maximum number of load steps to compute
BEG_LOAD	.1	Starting load factor
MAX_LOAD	1.2	Maximum load factor
MIN_LOAD	-1.0	Minimum load factor
MAX_ITERS	9	Maximum number of iterations per step
DES_ITERS	4	Desired number of iterations per step
NEWTON	<false>	Type of Newton-Raphson algorithm
FAC_STEPS	1	Number of steps between stiffness re-factorings
MAX_CUTS	3	Maximum number of step size cuts per step
CONV_CRITERIA	CHKCONV_E	Nonlinear convergence criteria (procedure name)
NL_TOL	0.001	Starting load step for nonlinear solution interval
PATH_SCALE	0.	Path (arclength) scale factor for re-starts
EXTRAPOLATE	<true>	Path extrapolation switch
NL_MATL	<false>	Material nonlinearity switch
NL_GEOM	2	Geometric nonlinearity level
COROTATION	<true>	Element corotation switch (for large rotations)
INITIALIZE	<true>	Initialization switch for re-starts in conjunction with adaptive mesh refinement.
LOAD_STIFF	<false>	
LINE_SEARCH	1.0	
ARCHIVE_STEP	10	

4.2.3 Argument Definitions

In this subsection, the procedure arguments summarized in Table 4.2-1 are defined in more detail. The arguments are listed alphabetically and many of the precise definitions are relegated to subordinate procedures and processors, where the actual options are determined. For example, the definition of REFINE_TECHNIQUE depends on which refinement processor the user selects via the REFINE_PROCESSOR argument, so the relevant options can be found in the corresponding refinement processor sections in Part III.

4.2.3.1 ASM_PROCESSOR Argument

This argument selects the matrix assembly processor to be used for assembling element (stiffness/mass) matrices into corresponding system matrices.

Argument syntax:

ASM_PROCESSOR = *asm_processor*

where *asm_processor* is the name of the matrix assembly processor. Current options include ASM (for h_t and h_c types of mesh refinement) and ASMs (for h_s mesh refinement only). (Default value: ASM)

4.2.3.2 AUTO_DOF_SUP Argument

This argument sets the automatic DOF (degree-of-freedom) suppression switch. This capability automatically suppresses extraneous DOFs, especially useful during adaptive mesh refinement. It is described in more detail in Section 2.10, *Automatic DOF Suppression and Drilling Stabilization.*

Argument syntax:

AUTO_DOF_SUP = *option* [, *angle_tol*]

where

Parameter	Description
option	Automatic DOF suppression option switch: {<true> | <false>}. If <true>, all DOFs (in the computational frame) that are unsupported by element stiffness will be suppressed throughout the adaptive refinement process. (Default value: <true>)
angle_tol	Angle tolerance to use for suppression of shell element drilling DOFs; see Section 2.10 for details. (Default value: depends on element type)

In most cases, it is best to leave the default setting intact.

4.2.3.3 AUTO_DRILL Argument

This argument sets the automatic drilling stiffness option. This option causes shell elements to add artificial drilling rotational stiffness to nodal DOFs that would otherwise be unstable computationally. See Section 2.10, *Automatic DOF Suppression and Drilling Stabilization*, and individual element processor sections in Chapter 7, *Element Processors*, for more information.

Argument syntax:

AUTO_DRILL = *option* [, *angle_tol* , *scale_fac*]

where

Parameter	Description
option	Automatic drilling stiffness switch: { <true> | <false> }. If <true>, certain shell element types will add artificial drilling stiffness to nodal DOFs that require stabilization. (Default value: <false>)
angle_tol	Angle tolerance to use for determining whether artificial drilling stiffness is needed at a given node. See Section 2.10 for details. (Default value: depends on element type)
scale_fac	Scale factor determining magnitude of artificial drilling stiffness to be added by selected shell elements. See individual element descriptions in Chapter 7 for interpretation. (Default value: depends on element type)

AUTO_DRILL is not recommended for nonlinear analysis.

4.2.3.4 AUTO_MPC Argument

This argument sets the automatic multi-point constraint (MPC) option for suppression of extraneous drilling DOFs, defined as rotations about the normal to a plate or shell element. Unless the element has intrinsic stiffness associated with such rotations, these DOFs may lead to a singular stiffness matrix. Turning the AUTO_MPC option on causes special constraints to be generated at nodes where insufficient drilling rotational stiffness is present, to suppress the rotation about the appropriate ("drilling") axis. This axis is generally not aligned with any of the computational axes, so the constraint will typically involve a linear combination of the rotational DOFs corresponding to the computational axes. See Section 2.10, *Automatic DOF Suppression and Drilling Stabilization*, for more information on this option and related options such as AUTO_DOF_SUP, AUTO_DRILL, and AUTO_TRIAD.

Argument syntax:

AUTO_MPC = *option* [, *angle_tol*]

where

Parameter	Description
option	Automatic multi-point constraint switch for drilling stabilization: { <true> | <false> }. If <true>, multi-dof constraints will be generated at nodes where drilling stabilization is needed. (Default value: <false>)
angle_tol	Angle tolerance to use for determining whether drilling stabilization is needed at a given node. See Section 2.10 for details. (Default value: depends on element type)

4.2.3.5 AUTO_TRIAD Argument

This argument sets the automatic computational triad (i.e., DOF direction) re-alignment option. This option, an alternative to AUTO_DRILL, causes re-alignment of the computational triads at all nodes that require drilling DOF stabilization as long as no boundary conditions have been defined there. The computational axes are re-aligned such that one of them is parallel to the average element surface-normal at the node. Then, extraneous (unstable) drilling rotational DOFs can be subsequently suppressed via the AUTO_DOF_SUP option. (See Section 2.10, *Automatic DOF Suppression and Drilling Stabilization*, for more information.)

Argument syntax:

AUTO_TRIAD = *option* [, *angle_tol*]

where

Parameter	Description
option	Automatic triad re-alignment option switch: { <true> | <false> }. If <true>, computational triads will be re-aligned with average element normal at all nodes that require drilling DOF stabilization, unless boundary conditions are defined there. (Default value: <false>)
angle_tol	Angle tolerance to use for determining whether drilling stabilization is needed at a given node. See Section 2.10 for details. (Default value: depends on element type)

AUTO_TRIAD should only be used in conjunction with AUTO_DOF_SUP. It cannot be used in conjunction with user-defined point forces and/or multi-point constraints.

4.2.3.6 BEG_LOAD Argument

This argument sets the starting load factor for nonlinear analysis.

Argument syntax:

BEG_LOAD = *beg_load*

See documentation for nonlinear solution procedures (Section 3.3, NL_STATIC_1) for details. (Default value: None)

4.2.3.7 BEG_MESH Argument

This argument sets the number of the first mesh to analyze at the start of the current AR run. The initial mesh is designated as mesh 0.

Argument syntax:

BEG_MESH = *beg_mesh*

where *beg_mesh* is the beginning mesh number. (Default value: 0)

4.2.3.8 BEG_STEP Argument

This argument sets the number of the first load step to be computed in a given nonlinear analysis interval.

Argument syntax:

BEG_STEP = *beg_step*

where *beg_step* is the beginning (or starting) step number. Initially, *beg_step* should be set equal to 1. For analysis re-start runs, *beg_step* should be set equal to the next step to compute (or re-compute). See documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for more details. (Default value: None)

4.2.3.9 BEG_STEP_REF Argument

This argument sets the first load step number at which adaptive mesh refinement can begin.

Argument syntax:

BEG_STEP_REF = *beg_step_ref*

where *beg_step_ref* is the beginning step number for mesh refinement. (Default value: 1)

4.2.3.10 CASE Argument

This argument sets the name of the case being analyzed. This name is used as the first part of all database file names associated with the case (e.g., *Case*.DBC, *Case*.DBE, ...). This name is typically the same as the model name used in Model Definition Procedures.

Argument syntax:

CASE = *Case*

where *Case* is the case name prefix in all associated database files. (Default: AR_TEST)

4.2.3.11 CONVERGE_TOL Argument

This argument sets the value of the adaptive mesh refinement (AR) global convergence tolerance. This is a relative error measure (in fractional form) below which convergence of the discrete solution to the governing equations is assumed and no further adaptive mesh refinement is performed. The quantitative interpretation of this error measure depends on the particular error estimation processor (ERR*i*) and refinement processor (REF1) selected (see ERROR_PROCESSOR and REF_PROCESSOR arguments).

Argument syntax:

CONVERGE_TOL = *converge_tol*

where *converge_tol* is the relative error tolerance in fractional form (e.g., .1 corresponds to 10 percent error). (Default value: .05)

4.2.3.12 COROTATION Argument

This argument selects the element corotational update option to be employed by the generic element processor for geometrically nonlinear analysis (i.e., large rotations, small strains).

Argument syntax:

COROTATION = *corotation*

where *corotation* may be set to 0 (off), 1 (medium), or 2 (high): Refer to the description of nonlinear solution procedures (Section 3.3, NL_STATIC_1) for more details. (Default value: 1)

4.2.3.13 DES_ITERS Argument

This argument sets the desired number of iterations for nonlinear convergence at each load step, which affects how the step size is adaptively updated during the run.

Argument syntax:

DES_ITERS = *des_iters*

where *des_iters* is the desired number of iterations. This is relevant only for nonlinear analysis. Refer to the documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for more details. (Default value: 4)

4.2.3.14 ELEMENT_GROUPS Argument

Provides a list of element group numbers to process during error estimation.

Argument syntax:

ELEMENT_GROUPS = *grp_1, grp_2, . . ., grp_NUM_GROUP*

where *grp_"i"* is a valid element group number and where *NUM_GROUP* is set via the NUM_GROUP argument. (Default value: 0, which implies ALL)

4.2.3.15 ERROR_FREQUENCY Argument

This argument sets the load step frequency at which spatial error estimation is performed.

Argument syntax:

ERROR_FREQUENCY = *error_frequency*

where *error_frequency* is the number of load steps between spatial error estimation. A value of 1 implies error are estimated at every step; a value of 0 implies no error estimation is to be performed. (Default value: 1)

4.2.3.16 ERROR_MEASURE Argument

This argument sets the name of the spatial error measure (e.g., strain_energy, mean_stress, ...) to be used within the error estimation processor selected via the ERROR_PROCESSOR argument. Error measure options are dependent on the error estimation processor, and some error processors may have only one option (in which case this argument is irrelevant).

Argument syntax:

ERROR_MEASURE = *error_measure*

where *error_measure* is the name of the error measure. (Default value: error estimation processor dependent)

4.2.3.17 ERROR_PROCESSOR Argument

This argument sets the name of the error estimation processor (ERR*i*) to be employed by AR_CONTROL (via the utility procedure, EST_ERR_1). See Chapter 10, *Error Estimation Processors*, for available options.

Argument syntax:

ERROR_PROCESSOR = *error_processor*

where *error_processor* is the name of the error estimation processor. Current options are summarized below.

error_processor	Description
ERR2	Smoothing-based error estimator a la Zienkiewicz (Default)
ERR4	Modified version of ERR2 by Levit, for built-up shell structures
ERR6	Modified version of ERR2

4.2.3.18 ERROR_TECHNIQUE Argument

This argument sets the name of the error estimation technique to be employed within the error estimation processor selected via the ERROR_PROCESSOR argument. Error estimation technique options are dependent on the error estimation processor, and some error processors may have only one option (in which case this argument is irrelevant).

Argument syntax:

ERROR_TECHNIQUE = *error_technique*

where *error_technique* is the name of the error estimation technique. (Default value: error estimation processor dependent)

4.2.3.19 EXTRAPOLATE Argument

This argument sets a flag determining whether or not to use quadratic extrapolation along a non-linear solution path to predict the load factor and displacement vector at the beginning of each load step.

Argument syntax:

EXTRAPOLATE = { <true> | <false> }

where <true> implies that quadratic extrapolation will be used. This is relevant only for nonlinear analysis. Refer to the documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for more details. (Default value: <true>)

4.2.3.20 FAC_STEPS Argument

This argument sets the number of load steps between stiffness matrix updates (i.e., re-forming and re-factoring) for nonlinear analysis.

Argument syntax:

FAC_STEPS = *fac_steps*

where *fac_steps* is a positive integer indicating that re-factoring of a new stiffness matrix will be performed every *fac_steps* load steps. This is relevant only for nonlinear analysis and only for the argument NEWTON=MODIFIED. Refer to the documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for more details. (Default value: 1)

4.2.3.21 FIXED_FRAME Argument

Sets an esoteric flag that is relevant only for h_s-refinement.

Argument syntax:

FIXED_FRAME = { <true> | <false> }

Do not change the default setting without the advice of a COMET-AR expert. (Default value: <false>)

4.2.3.22 LDI_C Argument

This argument sets the logical device index associated with the central COMET-AR database file, which must exist before calling AR_CONTROL; it is typically named *Case*.DBC, where *Case* is the case name.

Argument syntax:

LDI_C = *ldi_c*

where *ldi_c* is the logical device index (a positive integer) of the .DBC file. (Default value: 1)

4.2.3.23 LDI_E Argument

This argument sets the logical device index associated with the element matrix database file, typically named *Case*.DBE.

Argument syntax:

LDI_E = *ldi_e*

where *ldi_e* is the logical device index (a positive integer) of the *Case*.DBE file. If *ldi_e* is not equal to *ldi_c* (see the LDI_C argument) then all element matrices (e.g., stiffness and mass) for the current mesh will be stored on a separate *Case*.DBE file. If *ldi_e* = *ldi_c*, then all element matrices will be stored on the *Case*.DBC file, i.e., a separate *Case*.DBE file will not be created. (Default value: 2)

If a separate *Case*.DBE file is created, it will be deleted and re-created with each new adaptive mesh.

4.2.3.24 LDI_R Argument

This argument sets the logical device index associated with the selected results database file, typically named *Case*.DBR.

Argument syntax:

LDI_R = *ldi_r*

where *ldi_r* is the logical device index (a positive integer) of the .DBR file. This file will be used to store all user-selected displacement results (see arguments N_SELECT, SEL_NODES, and SEL_DOFS) as well as key solution parameters for nonlinear analysis. (Default value: 4)

To create a .DBR file separate from the .DBC file, it must be opened via an *OPEN *ldi_r* directive before invoking AR_CONTROL. Alternatively, if *ldi_r* = *ldi_c*, the selected results will be stored in the .DBC file.

4.2.3.25 LDI_S Argument

This argument sets the logical device index associated with the system matrix database file, typically named *Case*.DBS.

Argument syntax:

LDI_S = *ldi_s*

where *ldi_s* is the logical device index (a positive integer) of the *Case*.DBS file. If *ldi_s* is not equal to *ldi_c* (see LDI_C argument), then all system matrices (e.g., stiffness and mass) for the current mesh will be stored on a separate *Case*.DBS file. If *ldi_s* = *ldi_c*, then all system matrices will be stored on the *Case*.DBC file, i.e., a separate *Case*.DBS file will not be created. (Default value: 3)

If a separate *Case*.DBS file is created, it will be deleted and re-created with each new adaptive mesh.

4.2.3.26 MATRIX_UPDATE Argument

This argument sets the matrix-update mode for hierarchical adaptive refinement (relevant only for h_s-refinement).

Argument syntax:

MATRIX_UPDATE = { FULL | PARTIAL }

where FULL implies that the entire stiffness matrix is reformed for each new mesh, and where PARTIAL implies that only the updated mesh contributions to the stiffness matrix are reformed for each new mesh. (Default value: FULL)

4.2.3.27 MAX_ASPECT_RATIO Argument

This argument sets the maximum element aspect ratios before and after prospective adaptive mesh refinement.

Argument syntax:

MAX_ASPECT_RATIO = *before, after*

where *before* denotes the maximum element aspect ratio before a prospective mesh refinement, and *after* denotes the maximum element aspect ratio after a prospective mesh refinement. If either of these limits would be violated, an alternate element refinement pattern is selected. This is relevant primarily for transition-based (h_t) refinement, where aspect ratios can be used to control the

degree of element distortion. See Chapter 11, *Mesh Refinement Processors*, for more information. (Default value: 0,0)

4.2.3.28 MAX_CUTS Argument

This argument sets the maximum number of step cuts allowed during the current nonlinear analysis run.

Argument syntax:

MAX_CUTS = *max_cuts*

where *max_cuts* is the maximum number of cuts allowed. A step cut refers to a halving of the load, or arclength, step size used to advance the solution from one step to the next. Step cuts are performed only if the maximum number of iterations (specified via the MAX_ITERS argument) is exceeded without converging at a given step. This is relevant only for nonlinear analysis. Refer to documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for details. (Default value: 3)

4.2.3.29 MAX_H_LEVEL Argument

This argument sets the maximum number of levels of adaptive *h*-refinement allowed within any one element. If the mesh refinement processor (REF*i*) determines that more than this many levels of *h*-refinement are necessary to achieve convergence, the adaptive analysis is terminated.

Argument syntax:

MAX_H_LEVEL = *max_h_level*

where *max_h_level* denotes the maximum number of levels of *h*-refinement permitted by the user. for any one element. See Chapter 11, *Mesh Refinement Processors*, for more information. (Default value: 10)

4.2.3.30 MAX_ITERS Argument

This argument sets the maximum number of iterations allowed for nonlinear convergence at a given load step.

Argument syntax:

MAX_ITERS = *max_iters*

where *max_iters* is the maximum number of iterations allowed. If *max_iters* iterations have been performed at a given step, and nonlinear convergence (to an equilibrium state) has not yet been

obtained, the nonlinear solution procedure will attempt to cut the step size as many times as allowed by the argument MAX_CUTS. If the limits set by both MAX_CUTS and MAX_ITERS have been reached, then the run will be terminated and the user will have to try a different strategy. This is relevant only for nonlinear analysis. Refer to documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for details. (Default value: 9)

4.2.3.31 MAX_LOAD Argument

This argument sets the maximum load factor for the nonlinear analysis.

Argument syntax:

MAX_LOAD = *max_load*

where *max_load* is the applied load factor beyond which the nonlinear analysis is terminated. This is relevant only for nonlinear analysis. Refer to documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for details. (Default value: None)

4.2.3.32 MAX_MESHES Argument

This argument sets the maximum number of meshes to analyze within the current run. The highest potential mesh number for the current run is equal to BEG_MESH+MAX_MESHES–1; thus, the maximum number of adaptive mesh updates for the run is simply MAX_MESHES–1.

Argument syntax:

MAX_MESHES = *max_meshes*

where *max_meshes* is the maximum number of meshes to analyze. (Default value: 1)

4.2.3.33 MAX_MESH_STEP Argument

This argument sets the maximum allowable number of mesh iterations per step to perform in a nonlinear analysis with adaptive mesh refinement.

Argument syntax:

MAX_MESH_STEP = *max_mesh_step*

where *max_mesh_step* is the maximum number of meshes per step. This value may be superceded by MAX_MESHES, which is the maximum number of total meshes allowed per run. (Default value: 5)

4.2.3.34 MAX_P_LEVEL Argument

This argument sets the maximum number of levels of uniform *p*-refinement allowed for the model. If the mesh refinement processor (REF*i*) determines that more than this many levels of *p*-refinement are necessary to achieve convergence, the adaptive analysis is terminated.

Argument syntax:

MAX_P_LEVEL = *max_p_level*

where *max_p_level* denotes the maximum number of levels of uniform *p*-refinement permitted. See Chapter 11, *Mesh Refinement Processors*, for more information. (Default value: 5)

4.2.3.35 MAX_STEPS Argument

This argument sets the maximum number of load steps to compute during the current nonlinear analysis run.

Argument syntax:

MAX_STEPS = *max_steps*

where *max_steps* is the maximum number of steps to compute in the current run, not to be confused with the number of the highest load step in the analysis. This is relevant only for nonlinear analysis. Refer to documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for details. (Default value: None)

4.2.3.36 MIN_LOAD Argument

This argument sets the minimum load factor for a nonlinear analysis.

Argument syntax:

MIN_LOAD = *min_load*

where *min_load* establishes a lower limit on the applied load factor, which should be less than the starting load factor specified by the BEG_LOAD argument. This is relevant only for nonlinear analysis. Refer to documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for details. (Default value: None)

4.2.3.37 MTX_BUFFER_SIZE Argument

This argument sets the size of the memory buffer to be used for matrix factorization and solution by certain matrix solution processors.

Argument syntax:

MTX_BUFFER_SIZE = *mtx_buffer_size*

where *mtx_buffer_size* is the size of the buffer in terms of logical variables. (Default value: 500000)

4.2.3.38 NEWTON Argument

This argument determines the type of Newton-Raphson iteration algorithm to use for nonlinear analysis.

Argument syntax:

NEWTON = { MODIFIED | TRUE }

If NEWTON=MODIFIED, stiffness matrix updates (re-forming and re-factoring) will be performed only at the beginning of every *fac_steps* load steps, where *fac_steps* is set via the FAC_STEPS argument. If NEWTON=TRUE, stiffness matrix updates will be performed at each iteration of every *fac_steps* load steps. This is relevant only for nonlinear analysis. Refer to documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for details. (Default value: MODIFIED)

4.2.3.39 NL_CONV_CRITERIA

This argument selects the command-language procedure to be used to assess nonlinear convergence at each iteration of a nonlinear analysis.

Argument syntax:

NL_CONV_CRITERIA = *procedure_name*

where *procedure_name* is the name of the convergence-checking procedure. This is relevant only for nonlinear analysis. See documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for more details on options. (Default value: CHKCONV_E (energy norm))

4.2.3.40 NL_GEOM Argument

This argument selects the geometrical nonlinearity option to be used in the current analysis.

Argument syntax:

NL_GEOM = *nl_geom*

where *nl_geom* is the option number, and may be set to 0 (geometrically linear), 1 (geometrically nonlinear, but linear element strain-displacement relations), or 2 (geometrically nonlinear including nonlinear element strain-displacement relations). This is relevant only for nonlinear analysis. Refer to documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for details. (Default value: 0)

4.2.3.41 NL_MATL Argument

This argument selects the material nonlinearity option to be used in the current analysis run.

Argument syntax:

NL_MATL = *nl_matl*

where *nl_matl* may be set to 0 (materially linear) or 1 (materially nonlinear). This is relevant only for nonlinear analysis. Refer to documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1)for details. (Default value: 0)

4.2.3.42 NL_TOL Argument

This argument sets the value of the error tolerance used to establish convergence of the nonlinear equilibrium iteration process at each load step.

Argument syntax:

NL_TOL = *nl_tol*

where *nl_tol* is the error tolerance in the error norm specified by NL_CONV_CRITERIA. This is relevant only for nonlinear analysis. Refer to documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for details. (Default value: 1.e-3)

4.2.3.43 NUM_GROUP Argument

This argument sets the number of element groups to be processed during error estimation. If NUM_GROUPS > 0, a corresponding list of element group numbers may be set via the ELEMENT_GROUP argument.

Argument syntax:

NUM_GROUPS = *num_group*

where *num_group* is the number of element groups to process. (Default value: 0 which implies ALL)

4.2.3.44 NUM_REFINE_TOLS Argument

This argument sets the number of local (element) error tolerances that will be used to guide adaptive refinement. The REFINE_TOLS argument specifies the error values for these tolerances, and the REFINE_LEVELS argument indicates the number of levels of refinement to perform when each tolerance is exceeded.

Argument syntax:

NUM_REFINE_TOLS = *num_refine_tols*

where *num_refine_tols* denotes the number of refinement tolerances. See Chapter 11, *Mesh Refinement Processors*, for more information. (Default value: 1)

4.2.3.45 NUM_STEP_REF Argument

This argument sets the number of nonlinear load steps between adaptive mesh refinement intervals. It is not relevant for linear static analysis.

Argument syntax:

NUM_STEP_REF = *num_step_ref*

where *num_step_ref* is the number of load steps between adaptive mesh refinement intervals. For example, if *num_step_ref* is 1, adaptive mesh refinement will be performed at every step; if it is 2, at every other step, but only if dictated by spatial error estimates. (Default value: 1)

4.2.3.46 N_SELECT Argument

This argument specifies the number of user-selected displacement components to be saved in the results database (see argument LDI_R) for nonlinear analysis.

Argument syntax:

N_SELECT = *n_select*

where *n_select* is the number of displacement components to save. The actual node and DOF numbers identifying these displacement components are specified via the SEL_NODES and SEL_DOFS arguments. (Default value: 0)

4.2.3.47 OLD_MESH Argument

This argument sets the number of the mesh from which to restart an adaptive analysis. If BEG_MESH=0, this argument is irrelevant. If BEG_MESH > 0, the default is MAX(*beg_mesh* –

1,0). The main use of this argument is to allow mesh refinement to be repeated from some earlier mesh, but with different adaptive refinement parameters. Error estimates must already be available for the mesh specified by OLD_MESH in order to restart from that mesh.

Argument syntax:

OLD_MESH = *old_mesh*

where *old_mesh* denotes the number of the mesh from which to restart. The number of the first mesh to be computed (or recomputed) will therefore be *old_mesh*+1. See Chapter 11 for more information. (Default: MAX (*beg_mesh*–1, 0))

4.2.3.48 PATH_SCALE Argument

This argument sets a scale factor to be applied to the current arclength increment, Δl, for the first step in a nonlinear re-start run. It is thus a manual way to adjust the solution step size.

Argument syntax:

PATH_SCALE = *path_scale*

where *path_scale* is a non-negative floating point number. If *path_scale* is set to 1, the step-size from the previous step (i.e., *beg_step–1*, where *beg_step* is set by the BEG_STEP argument) will be used to compute the first new step (*beg_step*). The main function of this argument is for the user to override the procedure's step-size adjustment algorithm, in cases where the user has a better idea based on experience. This is relevant only for nonlinear analysis. Refer to documentation on nonlinear solution procedures (Section 3.3, NL_STATIC_1) for details. (Default value: 0 => automatic step-size adjustment algorithm will be used to make re-start mimic continuation without re-start)

4.2.3.49 POST Argument

This argument enables or disables a user-written post-processing procedure to be invoked by the nonlinear solution procedure.

Argument syntax:

POST = { <true> | <false> }

where <true> causes the user-written post-processing procedure to be invoked and <false> prevents it from being invoked. (Default value: <false>)

4.2.3.50 REFINE_DIRS Argument

Establishes a list of intrinsic element directions in which to allow adaptive refinement.

Argument syntax:

REFINE_DIRS = *dir1* [, *dir2* [, *dir3*]]

where *dir1*, *dir2*, and *dir3* are intrinsic element direction numbers (i.e., in the elements internal, or natural, coordinate system), and each may take on a value between 1 and the maximum number of intrinsic element dimensions (i.e., 3 for 3D elements, 2 for 2D elements and 1 for 1D elements). This can eliminate unnecessary refinement, for example, in axisymmetric shell problems where only one of the surface directions need be refined. See Chapter 11, *Mesh Refinement Processors*, for more information. (Default value: 1, 2, 3).

4.2.3.51 REFINE_INDICATOR Argument

This argument sets the type of element refinement indicator to be used by the adaptive refinement processor. The refinement indicator is the criterion used to determine whether an element's error estimate is high enough to warrant refinement. The values of the refinement indicator denoting various levels of refinement are set by the REFINE_TOLERANCES argument.

Argument syntax:

REFINE_INDICATOR = *refine_indicator*

where *refine_indicator* denotes the name of the element refinement indicator to be used. (Default value: AVE. See Chapter 11, *Mesh Refinement Processors*, for details.)

4.2.3.52 REFINE_LEVELS Argument

Sets an array of element refinement levels corresponding to the array of refinement tolerances specified via the REFINE_TOLS argument. An element refinement level is defined as one application of local refinement, employing the refinement type specified via the REFINE_TECHNIQUE argument (e.g., h_r, h_c, h_s or p).

Argument syntax:

REFINE_LEVELS = *ref_lev_1, ref_lev_2, ... ref_lev_NUM_REFINE_TOLS*

where *ref_lev_"i"* denotes the number of levels to refine an element when the element refinement (error) indicator exceeds the tolerance specified by *ref_tol_"i"* in the REFINE_TOLS argument; and NUM_REFINE_TOLS is the value set in the NUM_REFINE_TOLS argument. (See Chapter 11, *Mesh Refinement Processors*, for details.) (Default value: 1)

4.2.3.53 REFINE_PROCESSOR Argument

This argument sets the name of the mesh refinement processor (REF*i*) to be invoked by AR_CONTROL (via the REF_MESH_1 utility procedure).

Argument syntax:

REFINE_PROCESSOR = *refine_processor*

where *refine_processor* is the name of the mesh refinement processor. Current options are summarized below.

refine_processor	Description
REF1	Contains a variety of adaptive mesh refinement techniques (Default)

Consult Chapter 11, *Mesh Refinement Processors*, for more details.

4.2.3.54 REFINE_TECHNIQUE Argument

This argument sets the refinement technique to be employed by the mesh refinement processor (REF*i*) specified via the REFINE_PROCESSOR argument.

Argument syntax:

REFINE_TECHNIQUE = *refine_technique*

where *refine_technique* is the name of the refinement technique. For example, in conjunction with processor REF1, the REFINE_TECHNIQUE argument might be set equal to "ht", "hc", or "p" (among others). See Chapter 11, *Mesh Refinement Processors*, for details. (Default value: "hc")

4.2.3.55 REFINE_TOLS Argument

Sets an array of element refinement tolerances corresponding to the array of refinement levels specified via the REFINE_LEVELS argument. An element refinement tolerance is a limit in the value of the element error-based refinement indicator (see the REFINE_INDICATOR argument) beyond which an element is refined by a prescribed number of levels.

Argument syntax:

REFINE_TOLS = *ref_tol_1, ref_tol_2, ... ref_tol_NUM_REFINE_TOLS*

where *ref_tol_"i"* denotes the value of the element refinement indicator beyond which an element should be refined by *ref_lev_"i"* levels where *ref_lev_"i"* is specified in the REFINE_LEVELS argument; and NUM_REFINE_TOLS is the value set in the NUM_REFINE_TOLS argument. (See Chapter 11, *Mesh Refinement Processors*, for details.) (Default value: .05)

4.2.3.56 RENO_PROCESSOR Argument

This argument sets the name of the equation (or node) renumbering processor to be used in order to optimize matrix equation solving (time and/or storage).

Argument syntax:

RENO_PROCESSOR = *renumber_processor*

where *renumber_processor* is the processor name. Current options are summarized below.

renumber_processor	Description
RENO	Node renumbering using a geometric algorithm (Default)
RENOs	Node renumbering for h_s-refinement only
RSEQ	Node renumbering via a variety of order optimization algorithms

Consult the relevant sections in Chapter 6, *Pre-Processors*, for more details.

4.2.3.57 RENUMBER_OPT

This argument sets the equation renumbering option to use within the renumbering processor selected via the RENO_PROCESSOR argument (assuming RENUMBER=<true>).

Argument syntax:

RENUMBER_OPT = *renumber_option*

where *renumber_option* indicates the renumbering option and depends on the particular renumbering processor chosen. See processors RENO, RSEQ, etc., in Chapter 6. (Default value: 0)

4.2.3.58 SEL_DOFS Argument

This argument specifies a list of DOF numbers designating user-selected displacement components to be saved in the results database for nonlinear analysis. Each DOF number corresponds to

a node specified via the SEL_NODES argument. The total number of such nodal DOFS must equal to that specified via the N_SELECT argument.

Argument syntax:

SEL_DOFS = *dof(1), dof(2), . . ., dof(* N_SELECT*)*

where *dof(i)* represents a nodal DOF number (e.g., 1,2,3 typically denote the computational translations u_{xc}, u_{yc}, u_{zc}), and N_SELECT is set via the N_SELECT argument. See the SEL_NODES argument for correspondence. (Default value: 0)

4.2.3.59 SEL_NODES Argument

This argument specifies a list of node numbers designating user-selected displacement components to be saved in the results database for nonlinear analysis. Each node number corresponds to a DOF number specified via the SEL_DOFS argument. The total number of such node/DOF pairs must equal to that specified via the N_SELECT argument.

Argument syntax:

SEL_NODES = *node(1), node(2), . . ., node(* N_SELECT*)*

where *node(i)* represents a node number, and N_SELECT is set via the N_SELECT argument. See the SEL_DOFS argument for correspondence. (Default value: 0)

4.2.3.60 SKY_PROCESSOR Argument

Selects the matrix solution processor to be used for factoring and solving assembled linear equation systems.

Argument syntax:

SKY_PROCESSOR = *sky_processor*

where *sky_processor* is the name of the matrix solution processor. Current options are summarized below.

sky_processor	Description
SKY	Direct solution of skyline matrices by Gauss elimination (Default)
SKYs	Direct and/or iterative solution of skyline matrices in conjunction with h_s-refinement only
ITER	Iterative solution of compact matrices by PCG algorithm
PVSOLV	Direct solution of skyline matrices optimized for vector computers
VSS	Vectorized sparse solver (extremely fast and space-conserving)

Consult Chapter 12, *Matrix/Vector Processors*, for details on individual solution processors.

4.2.3.61 SMOOTH_PROCESSOR Argument

This argument selects the stress smoothing processor used in conjunction with error estimation.

Argument syntax:

SMOOTH_PROCESSOR = *smooth_processor*

where *smooth_processor* is the name of the stress smoothing processor. Current options are summarized below.

smooth_processor	Description
SMT	Smoothing processor based on Zienkiewicz smoothing algorithm
SMZ	Smoothing processor based on Tessler smoothing algorithm

Consult Chapter 9, *Smoothing Processors*, for more details. (Default: none)

If this argument is not set by the user, it is assumed that no smoothing processor is needed and that error estimation will be performed entirely by the error estimation processor selected via the ERROR_PROCESSOR argument. Conversely, if SMOOTH_PROCESSOR is set to one of the above options, then the user must select an error estimation processor that is capable of "post-processing" smoothed data to obtain error estimates such as ERRSM.

4.2.3.62 SMOOTH_LOCATIONS Argument

This argument specifies the locations at which smoothed data is to be computed and stored by the SMOOTH_PROCESSOR.

Argument syntax:

SMOOTH_LOCATIONS = { INTEG_PTS | NODES | CENTROIDS }

where INTEG_PTS refers to element integration points, NODES refers to element nodes, and CENTROIDS refers to element centroids. (Default value: INTEG_PTS)

4.2.3.63 SMOOTH_OPTIONS Argument

This argument sets any parameters required by the smoothing processor selected via the SMOOTH_PROCESSOR argument.

Argument syntax:

SMOOTH_OPTIONS = *smooth_options*

where *smooth_options* represents a list of options (i.e., parameters) dependent on the particular smoothing processor selected. Consult Chapter 9, *Smoothing Processors*, for details on what (if any) parameters are required here. Typically this argument is used to select non-default smoothing options. (Default value: 0)

4.2.3.64 SOLN_PROCEDURE Argument

This argument sets the name of the solution procedure to be employed by AR_CONTROL for solving the equations corresponding to a given mesh.

Argument syntax:

SOLN_PROCEDURE = *soln_procedure*

where *soln_procedure* is the name of the solution procedure. Current options are L_STATIC_1 (linear static analysis) or NL_STATIC_1 (nonlinear static analysis). (Default value: L_STATIC_1)

4.2.3.65 SOLVER_CONV_TOL Argument

This argument sets the convergence tolerance for the iterative linear equation solver if one has been selected via the SKY_PROCESSOR argument.

Argument syntax:

SOLVER_CONV_TOL = *solver_conv_tol*

where *solver_conv_tol* is the convergence tolerance. (Default value: 1.e-5)

4.2.3.66 SOLVER_MAX_ITER Argument

This argument sets the maximum number of iterations allowed by an iterative linear equation solver (e.g., ITER). This is relevant only if the SKY_PROCESSOR argument is set equal to the name of an iterative solver.

Argument syntax:

SOLVER_MAX_ITER = *solver_max_iter*

where *solver_max_iter* is the maximum number of iterations allowed. (Default value: 100)

4.2.3.67 STRESS Argument

This argument determines if and when element stresses, strains and strain energy densities are to be computed and stored (archived) in the database.

Argument syntax:

STRESS = { *stress_archival_frequency* }

where *stress_archival_frequency* indicates the number of load steps between stress archives. A value of 1 implies stresses will be archived at each step (or once for linear statics), and a value of <false> (or 0) implies that they will not be archived at all. (Default value: 1)

It is currently necessary to set *stress_archival_frequency* > 0 for all analyses involving adaptive mesh refinement

4.2.3.68 STR_DIRECTION Argument

This argument sets the stress/strain reference frame (x_s,y_s,z_s) for post-processing and/or error estimation purposes.

Argument syntax:

STR_DIRECTION = *str_direction*

where *str_direction* denotes the stress/strain direction. Current options are summarized below.

str_direction	Meaning
ELEMENT (or 0)	Express stress/strain components in the local element (integration point) reference frame (x_s=x_l, y_s=y_l, z_s= z_l). (Default)
GLOBAL { X | Y | Z }	Express stress/strain components in a permutation of the global reference frame, with x_s = x_g, y_g or z_g, if X, Y or Z is selected. For shell elements, the z_s direction is automatically aligned with the local element normal, z_l, direction.
{ 1 | 2 | 3 }	Same as GLOBAL {X | Y | Z } respectively.
FAB_DIR	Use the local fabrication axes for the stress frame, i.e., x_s=x_f, y_s=y_f, z_s=y_f. See Section 2.7, *Orientation of Fabrication Frames*.

4.2.3.69 STR_LOCATION Argument

This argument sets the element locations at which stresses, strains, and strain energy densities are computed for post-processing and/or error estimation purposes.

Argument syntax:

STR_LOCATION = *str_location*

where *str_location* denotes the stress/strain/energy locations. Current options are as shown below.

str_location	Meaning
INTEG_PTS	Element integration points (Default)
NODES	Element nodes
CENTROIDS	Element centroids

It is currently necessary to set STR_LOCATION = INTEG_PTS for all analyses involving adaptive mesh refinement

4.2.4 Database Input/Output Summary

A complete model definition database is required as input for the first run with AR_CONTROL (see Chapter 2, *Model Definition Procedures*). After the analysis, both solution data, as well as model definition data will have been output to the database for all meshes created and analyzed during the adaptive refinement iteration loop. The mesh index will appear as the third index in all dataset names. While most datasets will be stored in the main COMET-AR database file, *Case*.DBC, element and system matrices may be stored in the *Case*.DBE and *Case*.DBS files, depending on the user settings for the LDI_E and LDI_S arguments.

4.2.4.1 Input Datasets

Table 4.2-2 contains a list of datasets required (unless otherwise stated) as input by procedure AR_CONTROL. All of these datasets must be resident in the main COMET-AR database (*Case*.DBC, where *Case* is specified via the CASE argument). The datasets listed all correspond to the input mesh, *mesh*, which is set via the OLD_MESH argument and will be equal to 0 for the initial mesh. Also, *ldset* refers to the LOAD_SET argument and *conset* refers to the CONSTRAINT_SET argument.

Table 4.2-2 Input Datasets Required by Procedure AR_CONTROL

Dataset	File	Description
CSM.SUMMARY...*mesh*	*Case*.DBC	Model summary for input mesh
EltName.DEFINITION...*mesh*	*Case*.DBC	Element definition for input mesh
EltName.FABRICATION...*mesh*	*Case*.DBC	Element fabrication pointers for input mesh
EltName.GEOMETRY...*mesh*	*Case*.DBC	Element solid-model geometry for input mesh
EltName.INTERPOLATION...*mesh*	*Case*.DBC	Element interpolation data for input mesh
EltName.LOAD.*ldset*..*mesh*	*Case*.DBC	Element load definition for input mesh
NODAL.COORDINATE...*mesh*	*Case*.DBC	Nodal coordinates for input mesh
NODAL.DOF..*conset.mesh*	*Case*.DBC	Nodal DOF boundary condition codes for input mesh.
NODAL.TRANSFORMATION...*mesh*	*Case*.DBC	Nodal transformations between global and computational frames for input mesh
NODAL.SPEC_FORCE.*ldcase*..*mesh*	*Case*.DBC	Nodal specified forces for input mesh (optional)
NODAL.SPEC_DISP.*ldcase*..*mesh*	*Case*.DBC	Nodal specified displacements for input mesh (optional)

4.2.4.2 Output Datasets

Table 4.2-3 contains a list of datasets that may be created or updated in the database by procedure AR_CONTROL. Most of these datasets will be resident in the main COMET-AR database file (*Case*.DBC), but element and system matrices may be resident in the *Case*.DBE file and *Case*.DBS files, depending on the values of the user-specified arguments LDI_E and LDI_S. The datasets listed all correspond to the output mesh, *mo*, the newest mesh created and analyzed by procedure AR_CONTROL. The value of *mo* should be no greater than the value set by the END_MESH procedure argument. For linear analysis, result dataset names contain the load set (*ldset*) and constraint set (*conset*) numbers, while for nonlinear analysis these dataset names instead contain the load step (*step*) number. *ldset* and *conset* are set by the LOAD_SET and CONSTAINT_STEP argument.

> In addition to the current mesh, datasets for all of the intermediate meshes between the input mesh and the current mesh will be stored in the database by AR_CONTROL (.DBC file only).

Table 4.2-3 Output Datasets Created or Modified by Procedure AR_CONTROL

Dataset	File	Description
CSM.SUMMARY...*mesh*	*Case*.DBC	Model summary for output mesh
EltName.DEFINITION...*mesh*	*Case*.DBC	Element definition for output mesh
EltName.ERROR.*ldset.conset.mesh* or *EltName*.ERROR.*step..mesh*	*Case*.DBC	Element error estimates computed for output mesh
EltName.FABRICATION...*mesh*	*Case*.DBC	Element fabrication pointers for output mesh
EltName.GEOMETRY...*mesh*	*Case*.DBC	Element solid-model geometry for output mesh
EltName.INTERPOLATION...*mesh*	*Case*.DBC	Element interpolation data for output mesh
EltName.LOAD.*ldset.mesh*	*Case*.DBC	Element load definition for output mesh
EltName.REFINEMENT...*mesh*	*Case*.DBC	Element refinement table for output mesh
EltName.STIFFNESS...*mesh*	*Case*.DBE	Element matrices for output mesh
EltName.STRAIN.*ldset.conset.mesh* or *EltName*.STRAIN.*step..mesh*	*Case*.DBC	Element strains computed for output mesh
EltName.STRESS.*ldset.conset.mesh* or *EltName*.STRESS.*step..mesh*	*Case*.DBC	Element stresses computed for output mesh (and step if nonlinear)
EltName.STRAIN_ENERGY.*ldset.conset.mesh* or *EltName*.STRAIN_ENERGY.*step..mesh*	*Case*.DBC	Element strain energy densities computed for output mesh (and step if nonlinear)
NODAL.COORDINATE...*mesh*	*Case*.DBC	Nodal coordinates for output mesh
NODAL.DISPLACEMENT.*ldset.conset.mesh* or NODAL.DISPLACEMENT.*step..mesh*	*Case*.DBC	Nodal displacements computed for output mesh (and step if nonlinear)
NODAL.ORDER..*conset.mesh*	*Case*.DBC	Nodal re-ordering array, defined by node renumbering processor (optional)
NODAL.DOF..*conset.mesh*	*Case*.DBC	Nodal DOF boundary condition codes and equation numbers for output mesh
NODAL.ROTATION.*step.mesh*	*Case*.DBC	Nodal rotations for nonlinear analysis
NODAL.TRANSFORMATION...*mesh*	*Case*.DBC	Nodal transformations between global and computational frames for output mesh
NODAL.SPEC_FORCE.*ldset..mesh*	*Case*.DBC	Nodal specified forces for output mesh (optional)
NODAL.SPEC_DISP.*ldset..mesh*	*Case*.DBC	Nodal specified displacements for output mesh (optional)
LINE.REFINEMENT...*mesh*	*Case*.DBC	Line refinement table for output mesh
SURFACE.REFINEMENT...*mesh*	*Case*.DBC	Surface refinement table for output mesh (only if 3D elements present)
SYSTEM.STIFFNESS...*mesh*	*Case*.DBS	Assembled system stiffness matrix
SYSTEM.VECTOR...*mesh*	*Case*.DBS	System vector used to store assembled force and displacement vectors during equation solution process.

For details on the contents of any of the datasets in Table 4.2-3, refer to Chapter 15, *Database Summary.*

4.2.5 Subordinate Procedures and Processors

4.2.5.1 Subordinate Procedures

A list of COMET-AR procedures invoked directly by procedure AR_CONTROL is provided in Table 4.2-4. Documentation on these procedures may be found in Chapter 3, *Basic Solution Procedures*, and Chapter 5, *Utility Procedures.*

Table 4.2-4 Subordinate Procedures to Procedure AR_CONTROL

Procedure	Type	Function
L_STATIC_1	Solution	Performs linear static structural analysis
NL_STATIC_1	Solution	Performs nonlinear static structural analysis
EST_ERR_1	Utility	Performs error estimation via the error estimation processor (ERR*i*) selected by the user with the ERROR_PROCESSOR argument
EST_ERR_SM	Utility	Performs error estimation via combination of smoothing processor and error processor ERRSM
REF_MESH_1	Utility	Performs adaptive mesh refinement via the mesh refinement processor (REF*i*) selected by the user with the REFINE_PROCESSOR argument

4.2.5.2 Relevant Subordinate Processors

Table 4.2-5 lists COMET_AR processors that are invoked directly by procedure AR_CONTROL and user-specified processors that are invoked indirectly through any of the subordinate procedures listed in Table 4.2-4. Documentation on these processors may be found in the chapter on the corresponding processor type.

Table 4.2-5 Relevant Subordinate Processors to Procedure AR_CONTROL

Processor	Type	Function
Assembler	Matrix/Vector	Matrix assembly processor, selected via the ASM_PROCESSOR procedure argument
Renumberer	Pre-Processor	Equation/node renumbering processor, selected via the RENO_PROCESSOR procedure argument
Equation Solver	Matrix/Vector	Equation solver, set via the SKY_PROCESSOR argument

Table 4.2-5 Relevant Subordinate Processors to Procedure AR_CONTROL (Continued)

Processor	Type	Function
Smoother	Smoothing	Performs "stress" smoothing for smoothing-based spatial error estimation; set by the SMOOTH_PROCESSOR argument
ERR*i*	Error Estimation	Error estimation processor, selected via the ERROR_PROCESSOR procedure argument
REF*i*	Mesh Refinement	Mesh refinement processor, selected via the REFINE_PROCESSOR procedure argument
VEC	Matrix/Vector	Performs vector algebra for nonlinear solution procedures

4.2.6 Current Limitations

A summary of current limitations is given in Table 4.2-6.

Table 4.2-6 Current Limitations of Procedure AR_CONTROL

Limitation		Description	Work-Around
1	Adaptive Analysis Type	AR_CONTROL adaptive mesh refinement options have been tested predominantly for linear static analysis (i.e., with solution procedure L_STATIC_1). The extension to nonlinear analysis (via procedure NL_STATIC_1) is new and experimental.	If unsure about nonlinear adaptive capabilities, perform linear adaptive analysis first, then switch to nonlinear analysis starting from the refined mesh.
2	Robustness	AR_CONTROL is not fully automatic, nor is it foolproof. It has been used to develop and research adaptive mesh refinement techniques for aircraft shell structures, and much more work remains to be done before it can be considered a "robust" tool for production engineering.	The user should be prepared to intervene, by studying the technical report given in [1], the COMET-AR User's Tutorial [2], and various parts of this manual.
3	Error Estimates	In particular, the current error estimators may not be quantitatively accurate, even though they may be qualitatively acceptable and produce effective adaptive meshes.	Be conservative; e.g., choose an error tolerance of .02 (2%) if your actual target is .05 (5%).

4.2.7 Status and Error Messages

A summary of important status and error messages potentially printed by Procedure AR_CONTROL is given in Table 4.2-7.

Table 4.2-7 Status and Error Messages for Procedure AR_CONTROL

Status/Error Message		Potential Cause(s)	Suggested User Response
1	Adaptive refinement procedure converged	Error estimates indicate that the global error tolerance (specified via the CONVERGE_TOL argument) has been satisfied with the current mesh.	Celebrate, but not before examining critical solution quantities such as maximum stress and verifying convergence.
2	Adaptive refinement procedure terminated without convergence	After analyzing and creating meshes BEG_MESH through END_MESH, the user-specified global error tolerance has still not been satisfied. This is either because the current adaptive strategy requires more mesh iterations or it may be hung up on an intractable singularity such as a point force (which should not be employed with adaptive refinement).	Either restart from the current mesh and allow more mesh updates, or consider accepting the error levels already achieved.
3	Adaptive mesh refinement limits exceeded	After or during the current mesh refinement step a user-specified limit in problem size (e.g., MAX_H_LEVEL) has been exceeded while the convergence tolerance has not yet been met.	Increase the original limits, re-run the analysis with a different strategy (e.g., refinement technique and/or error tolerances), or accept the latest solution as the best available within budget.

4.2.8 Examples and Usage Guidelines

4.2.8.1 Example 1: New Linear Adaptive Analysis

```
*call AR_CONTROL (      CASE             =  AR_CASE_1     ; --
                        SOLN_PROCEDURE   =  L_STATIC_1    ; --
                        BEG_MESH         =  0             ; --
                        MAX_MESHES       =  4             ; --
                        ERROR_PROCESSOR  =  ERR4          ; --
                        ERROR_TECHNIQUE  =  SMOOTHING     ; --
                        REFINE_PROCESSOR =  REF1          ; --
                        REFINE_TECHNIQUE =  hc            ; --
                        REFINE_INDICATOR =  AVE           ; --
                        REFINE_TOLS      =  .05           ; --
                        CONVERGE_TOL     =  .05           ; --
                        MAX_H_LEVEL      =  5               )
```

In this example, an adaptive linear static analysis is requested, starting with mesh 0 (the initial model) and allowing up to 3 mesh updates (meshes 1, 2 and 3). For error estimation, processor ERR4 is requested to use smoothing-based error estimates (of element strain-energy densities). For mesh refinement, processor REF1 is requested to use h_c (constraint-based h) refinement, to employ the AVE element refinement indicator (which attempts to distribute element errors uniformly), to refine all elements whose relative errors are greater than 5% by one level of subdivi-

sion, and to terminate refinement when the global relative error is less than or equal to 5%, or when any element attempts to subdivide by more than 5 *h*-refinement levels. Many of the default values were explicitly used in this example for illustration purposes.

4.2.8.2 Example 2: Linear Adaptive Analysis Restart

```
*call AR_CONTROL (      CASE              =  AR_CASE_1      ; --
                        BEG_MESH          =  3              ; --
                        MAX_MESHES        =  3              ; --
                        ERROR_PROCESSOR   =  ERR4           ; --
                        ERROR_TECHNIQUE   =  SMOOTHING      ; --
                        REFINE_PROCESSOR  =  REF1           ; --
                        REFINE_TECHNIQUE  =  hc             ; --
                        REFINE_INDICATOR  =  AVE            ; --
                        REFINE_TOLS       = .05             ; --
                        CONVERGE_TOL      = .05             ; --
                        MAX_H_LEVEL       = 5                 )
```

This example is a sequel to Example 1, and assumes that the desired error convergence tolerance was not achieved via the first 3 mesh updates. The run invoked here will begin by performing error estimation and adaptive mesh refinement on mesh 3 and performing up to 2 more mesh updates. All of the other AR control parameters are identical to Example 1.

4.2.8.3 Example 3: NonLinear/NonAdaptive Analysis Initiation

```
*call AR_CONTROL (      CASE              =  AR_CASE_1      ; --
                        SOLN_PROCEDURE    =  NL_STATIC_1    ; --
                        BEG_STEP          =  1              ; --
                        MAX_STEPS         =  10             ; --
                        BEG_LOAD          =  .1             ; --
                        MAX_LOAD          =  1.0            ; --
                        NL_TOL            =  .00001           )
```

4.2.8.4 Example 4: NonLinear/Adaptive Analysis Initiation

```
*call AR_CONTROL (      CASE                =  AR_CASE_1        ; --
                        SOLN_PROCEDURE      =  NL_STATIC_1      ; --
                        BEG_STEP            =  1                ; --
                        MAX_STEPS           =  10               ; --
                        BEG_LOAD            =  .1               ; --
                        MAX_LOAD            =  1.0              ; --
                        NL_TOL              =  .00001           ; --
                        BEG_MESH            =  0                ; --
                        MAX_MESHES          =  5                ; --
                        ERROR_PROCESSOR     =  ERR2             ; --
                        ERROR_TECHNIQUE     =  SMOOTHING        ; --
                        REFINE_PROCESSOR    =  REF1             ; --
                        REFINE_TECHNIQUE    =  hc               ; --
                        REFINE_INDICATOR    =  AVE              ; --
                        REFINE_TOLS         =  .05              ; --
                        CONVERGE_TOL        =  .05                --
                                                                  )
```

4.2.8.5 Example 5: Linear Adaptive Analysis With Smoothing-based Error Estimation

```
*call AR_CONTROL (      CASE                =  AR_CASE_1        ; --
                        SOLN_PROCEDURE      =  L_STATIC_1       ; --
                        BEG_MESH            =  0                ; --
                        MAX_MESHES          =  4                ; --
                        SMOOTH_PROCESSOR    =  SMT              ; --
                        SMOOTH_LOCATIONS    =  INTEG_PTS        ; --
                        SMOOTH_OPTIONS      =  1.0              ; --
                        ERROR_PROCESSOR     =  ERRSM            ; --
                        ERROR_TECHNIQUE     =  SMOOTHING        ; --
                        REFINE_PROCESSOR    =  REF1             ; --
                        REFINE_TECHNIQUE    =  hc               ; --
                        REFINE_INDICATOR    =  AVE                )
                        REFINE_TOLS         =  .05
                        CONVERGE_TOL        =  .05
                        MAX_H_LEVEL         =  5
```

4.2.9 References

[1] Stanley, G., Levit, I., Hurlbut, B., and Stehlin, B., "Adaptive Refinement (AR) Strategies for Shell Structures; Part 1: Preliminary Research," *Preliminary NASA Contract Report*, 1991.

[2] Stehlin, B., "The COMET-AR User's Tutorial," *NASA Preliminary Contract Report*, February, 1993.

Chapter 5 Utility Procedures

5.1 Overview

This chapter describes existing COMET-AR command-language utility procedures that perform basic, low-level finite element analysis tasks. A section is dedicated to each of the currently available procedures which are listed in Table 5.1-1. They include a generic element procedure, generic solver procedures and generic adaptive refinement and error estimation procedures. These utility procedures may be invoked with a simple *CALL directive after running the COMET-AR macroprocessor (see Chapter 1).

Table 5.1-1 Outline of Chapter 5: Utility Procedures

Section	Procedure	Function
5.1	Overview	Introduction
5.2	ES	Performs various element tasks
5.3	EST_ERR_1	Performs error estimation
5.4	EST_ERR_SM	
5.5	FACTOR	Performs decomposition of a system matrix (Crout/Cholesky)
5.6	FORCE	Calculates force vectors, internal and external
5.7	INITIALIZE	Performs various initialization tasks
5.8	REF_MESH_1	Performs adaptive mesh refinement
5.9	SOLVE	Performs system equation solution
5.10	STIFFNESS	Computes element stiffness matrices and assembles the system stiffness matrix
5.11	STRESS	Performs stress recovery
5.12	MASS	

The above utility procedures invoke various COMET-AR processors as described in Part III.

5.2 Procedure ES

5.2.1 General Description

This procedure is a CLIP cover for the generic element processor, or ES (for Element/Structural), which provides a standard template for individual COMET-AR structural finite-element processors. These processors have names that begin with ES (e.g., ES1p, ES7p, ES36, ...). Each of these ES*i* processors performs all operations for all element types implemented within the processor, including the definition of element connectivity and loads during pre-processing, the formation of element force and stiffness arrays during the primary solution phase, and the formation of strains and stresses during the secondary solution phase of structural analysis.

This section describes the ES Utility Procedure, which automatically executes all element processors and types associated with a given model. For most analyses, users will not have to directly interact with the generic element (ES) processor or procedure except during model definition, where ES*i* processors are run directly (within model definition procedures) to define elements and element loads (with the DEFINE ELEMENTS and DEFINE LOADS commands). During the solution phase, element functions are automatically exercised via solution procedures and their subordinate utility procedures.

5.2.2 Argument Summary

Procedure ES may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 5.2-1.

Table 5.2-1 Procedure ES Input Arguments

Argument	Default Value	Description
COROTATION	<false>	Sets the default element corotational option
DISPLACEMENT	NODAL.DISPLACEMENT.1.1	Sets the default name of nodal displacement dataset
DRILL_STIFF	<false>	Sets the default value of artificial drilling stiffness parameter
DRILL_TOL	0	Sets the default value of drilling stabilization angle tolerance
FORCE	NODAL.FORCE.1.1	Sets the default name of nodal force dataset
FREEDOMS	ES.DOFS	
FUNCTION	—	Defines the function to be performed by the ES*i* processor
GCP	1	Sets the default *ldi* of GCP material and fabrication datasets

Table 5.2-1 Procedure ES Input Arguments (Continued)

Argument	Default Value	Description
LDI	1	Sets the default *ldi* of computational database library
LOAD_FACTOR	1.0	Sets the default load factor to be applied to element loads
LOAD_SET	1	Sets the default load set number for element loads
MASS	MASS	Sets the default name of output mass matrix dataset
MESH	0	Sets the mesh number
NL_GEOM	<false>	Sets the default geometric nonlinearity option
NL_LOAD	<false>	Sets the default load nonlinearity option
NL_MATL	<false>	Sets the default material nonlinearity option
NUM_CON_SETS	1	
PROJECTION	<false>	Sets the default element projection option
ROTATION	NODAL.ROTATION.1.1	Sets the default name of nodal rotation pseudovector dataset
SE_TOT	<false>	
STEP	0	Sets/resets load- or time-step number
STIFFNESS	STIFFNESS	Sets the default name of element stiffness dataset
STRAIN	—	Sets the default name of element strain dataset
STRAIN_ENERGY	—	Sets the default name of element strain energy dataset
STRESS	—	Sets the default name of element stress dataset
STR_DIRECTION	0	Sets the default stress/strain output coordinate system
STR_LOCATION	INTEG_PTS	Sets the default stress/strain output locations

5.2.3 Argument Definitions

In this subsection, the procedure arguments summarized in Table 5.2-1 are defined in more detail. The arguments are listed alphabetically. Refer to Chapter 7, *Element Processors*, for more details on the options.

5.2.3.1 COROTATION Argument

This argument sets the default element corotational option for geometrically nonlinear analysis. The corotational capability is built in to the generic element processor (ES) and enables beam and shell elements to be employed with arbitrarily large rotations (but small to moderate strains) even if the element strain-displacement relations do not intrinsically account for large rotations exactly.

Argument syntax:

COROTATION = *corotation_option*

where

corotation_option	Description
0 or <false>	Element corotation will not be used. (Default)
1	Basic element corotation will be used. This option is sufficient unless True-Newton iteration is begin performed at the nonlinear solution procedure level.
2	Higher-order element corotation will be used. This option should be used only if True-Newton iteration has been selected at the nonlinear solution procedure level, and even then may provide only marginal improvement in nonlinear convergence over option 1. It adds additional terms to the tangent stiffness matrix that render it more consistent.

5.2.3.2 DISPLACEMENT Argument

This argument changes the default *ldi* and name of the nodal displacement dataset.

Argument syntax:

DISPLACEMENT = *ds_name*

where *ds_name* is the nodal displacement dataset name.
(Default value: NODAL.DISPLACEMENT.1.1)

5.2.3.3 DRILL_STIFF Argument

This argument changes the default artificial drilling rotational stiffness option for (certain) shell element types.

Argument syntax:

DRILL_STIFF = *Option* [, *scale*]

where *Option* is either <true> or <false>, and *scale* is an integer scale factor that depends on the particular element type. (Default value: <false>)

5.2.3.4 DRILL_TOL Argument

This argument changes the default artificial drilling tolerance option for (certain) shell element types.

Argument syntax:

DRILL_TOL = *angle*

where *angle* is an integer angle tolerance indicating when some form of stabilization is required for shell element drilling rotational freedoms. If the angle between the shell-element normal and the average element normal (or a computational axis) at a node is less than this value, drilling stabilization may be required (depending on the element type). (Default value: 0)

5.2.3.5 FORCE Argument

This argument changes the default name of the nodal force dataset.

Argument syntax:

FORCE = *ds_name*

where *ds_name* is the new dataset name. (Default value: NODAL.FORCE.1.1)

5.2.3.6 FUNCTION Argument

This argument defines the function to be performed by the ES*i* processor.

Argument syntax:

FUNCTION = *function*

where

Function	Description
INITIALIZE	Creation of element INTERPOLATION datasets, element AUX_STORAGE datasets, and initialization of constitutive datasets
FORM FORCE	Forms element force vectors (internal, external, or residual)
FORM STIFFNESS	Forms element stiffness matrices (material, geometric, load, or tangent)
FORM MASS	Forms element mass matrices (consistent or lumped)
FORM STRAIN	Computes element strains
FORM STRAIN_ENERGY	Computes element strain energy
FORM STRESS	Computes element stresses

(Default value: None)

5.2.3.7 GCP Argument

This argument changes the default database logical device index (*ldi*) associated with all datasets managed by the Generic Constitutive Processor.

Argument syntax:

GCP = *gcp_ldi*

where *gcp_ldi* is the logical device index. (Default value: 1)

5.2.3.8 LOAD_FACTOR Argument

This argument changes the default load factor to be applied to all element loads.

Argument syntax:

LOAD_FACTOR = *load_factor*

where *load_factor* is a floating-point scale factor. (Default value: 1.0)

5.2.3.9 LOAD_SET Argument

This argument changes the default load set number for element loads during either load definition or consistent external force formation.

Argument syntax:

LOAD_SET = *load_set*

where *load_set* is an integer load-set number. (Default value: 1)

5.2.3.10 LDI Argument

This argument changes the default logical device index (*ldi*) for all datasets input/output by the current ES*i* processor, except those for which an explicit *ldi* is used in a separate database command (e.g., STIFFNESS or GCP_LDI).

Argument syntax:

LDI = *ldi*

where *ldi* is the logical device index of the database library. (Default value: 1)

5.2.3.11 MASS Argument

This argument changes the default name of the element (consistent) or nodal (lumped) mass datasets.

Argument syntax:

MASS = *ds_name*

where *ds_name* is the new dataset name. (Default value: MASS)

5.2.3.12 MESH Argument

This argument changes the default mesh number used in all dataset names (unless otherwise specified via a separate dataset command).

Argument syntax:

MESH = *mesh*

where *mesh* is an integer number, typically set to the current mesh number. (Default value: 0)

5.2.3.13 NL_GEOM Argument

This argument changes the default geometric nonlinearity option. It is often used in conjunction with the COROTATION command.

Argument syntax:

NL_GEOM = *nl_geom_option*

where

nl_geom_option	**Description**
0 or <false>	The analysis is geometrically linear; linear element strain-displacement relations will be employed and element corotational will be disregarded. (Default)
1	The analysis is geometrically nonlinear, but only linear element strain-displacement relations will be used. With this option, geometric nonlinearity must be accounted for via element corotation, which for many beam/shell element types is not as accurate as option 2.
2	The analysis is geometrically nonlinear, and nonlinear element strain-displacement relations will be used. Element corotation may or not be selected with this option. For many beam/shell element types, nonlinear element strain-displacement relations enhances corotation, making it more accurate for a given mesh and rotation magnitude.

5.2.3.14 NL_LOAD Argument

This argument changes the default load nonlinearity option. It affects whether "live" loads are to be processed as part of the external force vector or the tangent stiffness matrix.

Argument syntax:

NL_LOAD = *nl_load_option*

where

nl_load_option	Description
0 or <false>	Ignore load nonlinearity (i.e., displacement dependence). Only displacement-independent ("dead") external loads are to be processed in the following FORM FORCE or FORM STIFFNESS command. (Default)
1	Include load nonlinearity. Only displacement-dependent ("live") external loads are to be processed in the following FORM FORCE or FORM STIFFNESS command.

5.2.3.15 NL_MATL Argument

This argument changes the default material nonlinearity option.

Argument syntax:

NL_MATL = *nl_matl_option*

where

nl_matl_option	Description
0 or <false>)	The analysis is materially linear; ignore nonlinearity in any material constitutive models. (Default)
1	The analysis is materially nonlinear, include nonlinearity in material constitutive models if it exists.

5.2.3.16 PROJECTION Argument

This argument changes the default element "rigid-body projection" option. The rigid-body projection option is the linearized counterpart of the corotation option and modifies the stiffness matrix and displacement vector so that they are free from spurious strains due to (infinitesimal) rigid-body motion. This is relevant only for elements that do not preserve rigid-body modes exactly (for example, warping-sensitive shell elements such as those in processor ES5) and can make a difference in both linear and nonlinear analysis.

Argument syntax:

PROJECTION = *projection_option*

where

projection_option	Description
0 or <false>	Element rigid-body projection will not be performed. (Default)
1	Element rigid-body projection will be performed.

5.2.3.17 ROTATION Argument

This argument changes the default name of the nodal rotation (pseudovector) dataset.

Argument syntax:

ROTATION = *ds_name*

where *ds_name* is the new dataset name. (Default value: NODAL.ROTATION.1.1)

5.2.3.18 STEP Argument

This argument changes the default load- or time-step number used in many solution dataset names (unless otherwise specified via a separate dataset command).

Argument syntax:

STEP = *step*

where *step* is an integer number, typically set to the current step number. (Default value: 0)

5.2.3.19 STIFFNESS Argument

This argument changes the default name of the element stiffness matrix dataset.

Argument syntax:

STIFFNESS = *ds_name*

where *ds_name* is the new dataset name. (Default value: STIFFNESS)

5.2.3.20 STRAIN Argument

This argument changes the default name of the element strain dataset before using the FORM STRAIN command. It also causes strains to be output to the database by the FORM STRESS, FORM FORCE/INT, or FORM FORCE/RES FUNCTION arguments.

Argument syntax:

STRAIN = *ds_name*

where *ds_name* is the new dataset name. (Default value: None)

5.2.3.21 STRAIN_ENERGY Argument

This argument changes the default name of the element strain-energy density dataset before using the FORM STRAIN_ENERGY command. It also causes strain-energy densities to be output to the database by the FORM STRESS, FORM FORCE/RES, or FORM FORCE/INT FUNCTION arguments.

Argument syntax:

STRAIN_ENERGY = *ds_name*

where *ds_name* is the new dataset name. (Default value: None)

5.2.3.22 STRESS Argument

This argument changes the default *ldi* and name of the element stress dataset before using the FORM STRESS command. It also causes strains to be output to the database by the FORM FORCE/INT or FORM FORCE/RES FUNCTION arguments.

Argument syntax:

STRESS = *ds_name*

where *ds_name* is the new dataset name. (Default value: None)

5.2.3.23 STR_DIRECTION Argument

This argument changes the default stress or strain direction option prior to use of the FORM STRAIN, FORM STRESS, FORM FORCE/INT, or FORM FORCE/RES FUNCTION arguments.

Argument syntax:

STR_DIRECTION = *str_direction*

where

str_direction	Description
ELEMENT or 0	Use element local (integration point) coordinate system, x_l, y_l, z_l, as stress/strain output system: x_s, y_s, z_s. (Default)
GLOBAL { X | Y | Z }	The stress/strain output x_s axis is parallel to the global x_g, y_g, or z_g axis if X, Y or Z, respectively, is used in the subcommand. The stress/strain output z_s axis is parallel to the local element normal axis for shell elements, otherwise it is obtained by permuting the global axes. The stress/strain output y_s axis is defined by the right-hand-rule.
FAB_DIR	Use the local material-fabrication coordinate system, x_f, y_f, z_f, as the stress/strain output system, x_s, y_s, z_s.

5.2.3.24 STR_LOC Argument

This argument changes the default stress, strain or strain-energy location option prior to use of the FORM STRAIN, FORM STRESS, FORM STRAIN_ENERGY, FORM FORCE/INT, or FORM FORCE/RES FUNCTION arguments.

Argument syntax:

STR_LOC = *str_location*

where

str_location	Description
INTEG_PTS	Element stresses, strains, or strain-energy densities will be evaluated and stored at element integration points in the STR attribute of the specified EST dataset. (Default)
NODES	Element stresses, strains, or strain-energy densities will be evaluated at integration points, then extrapolated and stored at element nodes in the STRNOD attribute of the specified EST dataset.
CENTROIDS	Element stresses, strains, or strain-energy densities will first be evaluated at the element integration points, then averaged and stored at element centroids in the STRCEN attribute of the specified EST dataset. (If one of the element's integration points coincides with the centroid, the value computed there will be output rather than an average integration-point value.)

5.2.4 Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the ES*i* processor being used and the FUNCTION argument. These dataset requirements are documented in detail in Chapter 7, *Element Processors*.

5.2.5 Current Limitations

ES is a general purpose procedure and the only limitations on its usage are dictated by the limitations of the ES*i* processor being employed. Refer to individual ES*i* processors in Chapter 7 for specific processor limitations.

5.2.6 Status and Error Messages

ES does not print any status or error messages directly. All messages will be produces by the ES*i* processor being employed. Refer to individual ES*i* processors in Chapter 7 for specific processor messages.

5.2.7 Examples and Usage Guidelines

5.2.7.1 Example 1: Stiffness Matrix Formation

```
*call ES (      STIFFNESS           =  MATL_STIFF          ; --
                GCP                 =  4                   ; --
                NL_MATL             =  <false>             ; --
                NL_GEOM             =  <false>             ; --
                COROTATION          =  <false>             ; --
                PROJECTION          =  <false>             ; --
                MESH                =  3                   ; --
                FUNCTION            =  FORM STIFFNESS       )
```

In this example, the formation of element linear material stiffnesses is requested for mesh 3. The Generic Constitutive Processor database is stored in logical device index 4 and the element stiffness matrices will be stored in 1, *EltNam*.STIFFNESS...*mesh*.

5.2.8 References

None.

5.3 Procedure EST_ERR_1

5.3.1 General Description

Procedure EST_ERR_1 is a utility procedure for performing finite element solution error estimation. It automatically invokes the appropriate error estimation processor (see Chapter 10 for details).

Procedure EST_ERR_1 is typically invoked automatically by solution procedure AR_CONTROL during analyses with adaptive mesh refinement.

5.3.2 Argument Summary

Procedure EST_ERR_1 may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 5.3-1.

Table 5.3-1 Procedure EST_ERR_1 Input Arguments

Argument	Default Value	Description
ACCUMULATE	<false>	Accumulation of errors when processing by group switch
CONSTRAINT_SET	1	Specifies constraint-set number for error estimation
ERROR_MEASURE	STRAIN-ENERGY	
ERROR_PROCESSOR	ERR2	Name of error estimation processor to invoke
ERROR_TECHNIQUE	S	Error estimation technique (S => Smoothing)
GROUP	0	List of element groups for error estimation
LDI	1	Logical unit for computational COMET-AR database file (*Case*.DBC)
LOAD_SET	1	Specifies load-set number for error estimation
MESH	0	Specifies mesh number for error estimation
NUM_GROUP	0	Number of element groups for error estimation
STEP	0	Specifies load/time-step number for error estimation

5.3.3 Argument Definitions

In this subsection, the procedure arguments summarized in Table 5.3-1 are defined in more detail. The arguments are listed alphabetically and many of the precise definitions are relegated to subordinate procedures and processors, where the actual options are determined. For example, the definition of REFINE_TECHNIQUE depends on which refinement processor the user selects via the REFINE_PROCESSOR argument, thus the user is referred to the corresponding refinement processor section in Part III for details on the options.

5.3.3.1 ACCUMULATE Argument

This argument sets the error accumulation switch.

Argument syntax:

ACCUMULATE = *switch*

where *switch* is a flag instructing the ES procedure to run the ERRa processor after estimating all element groups errors to accumulate the total model errors (see Section 10.6).

5.3.3.2 CONSTRAINT_Argument

This argument defines the constraint set number associated with the element solution data for which error estimates are to be computed. This number should appear as the second cycle number in names of all element solution datasets, e.g., STRESS, STRAIN, and STRAIN_ENERGY (relevant only for linear static analysis).

Argument syntax:

CONSTRAINT_= *constraint_set*

where:

Parameter	Description
constraint_set	Constraint set number (Default value: 1)

5.3.3.3 ERROR_PROCESSOR Argument

This argument defines the error processor to be used for estimating the solution errors, e.g., ERR2, ERR4, or ERR6.

Argument syntax:

ERROR_PROCESSOR = *error_processor*

where *error_processor* is the name of the error estimation processor. (Default value: ERR2)

5.3.3.4 GROUP Argument

This argument defines the element group identity numbers for a group of elements that need to be processed by the ERR*i* processors for each of the element types specified.

Argument syntax:

GROUP = *first:last:incr*
or
GROUP = $g_1, g_2, \ldots, g_N$

where:

Parameter	Description
first	First group ID to be processed (Default value: 0; all groups)
last	Last group ID
incr	Group ID increment
g_i	Group ID

5.3.3.5 LDI Argument

This argument defines the logical device index for the computational database.

Argument syntax:

LDI = *ldi*

where:

Parameter	Description
ldi	Logical device index. (Default value: 1)

5.3.3.6 LOAD_SET Argument

This argument defines the load set number associated with the element solution data for which error estimates are to be computed. This number appears as the first cycle number in names of all element solution datasets, e.g., STRESS, STRAIN, and STRAIN_ENERGY (relevant only for linear static analysis).

Argument syntax:

LOAD_SET= *load_set*

where:

Parameter	Description
load_set	Load set number. (Default value: 1)

5.3.3.7 MESH Argument

This argument defines the mesh number associated with the model and solution data for which error estimates are to be computed. This number should appear as the third cycle number in names of all datasets, e.g., *EltNam*.ERROR.*ldset.conset.mesh*.

Argument syntax:

MESH = *mesh*

where:

Parameter	Description
mesh	Mesh number to be processed. (Default value: 0)

5.3.3.8 STEP Argument

This argument defines the solution step number associated with the element solution data for which error estimates are to be computed. This number appears as the first cycle number in names of all element solution datasets, e.g., STRESS, STRAIN, and STRAIN_ENERGY (relevant only for linear static analysis).

Argument syntax:

STEP = *step*

where:

Parameter	Description
step	solution step number. (Default value: None)

5.3.4 Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the ERR*i* processor being used. These dataset requirements are documented in detail in Chapter 10.

5.3.5 Current Limitations

EST_ERR_1 is a general purpose procedure and the only limitations on its use are dictated by the limitations of the ERR*i* processor being employed. Refer to individual ERR*i* processors in Chapter 10 for specific processor limitations.

5.3.6 Status and Error Messages

EST_ERR_1 does not print any status or error messages directly. All messages will be produced by the ERR*i* processor being employed. Refer to individual ERR*i* processors in Chapter 10 for specific processor messages.

5.3.7 Examples and Usage Guidelines

5.3.7.1 Example 1: ERROR Estimation Without Group Partition

```
*CALL EST_ERR_1 (      ERROR_PROCESSOR      =  ERR2              ; --
                       ERROR_TECNIQUE       =  S/BARLOW          ; --
                       MESH                 =  2                   )
```

In this example, error estimation processor ERR2 using the Zienkiewicz-Zhu global smoothing algorithm and Barlow point stress data will be employed for estimating the errors in mesh 2.

5.3.7.2 Example 2: ERROR Estimation With Group Partition

```
*CALL EST_ERR_1 (      ERROR_PROCESSOR      =  ERR6              ; --
                       ERROR_TECNIQUE       =  S/BARLOW          ; --
                       MESH                 =  1                 ; --
                       NUM_GROUP            =  2                 ; --
                       GROUP                =  1, 2              ; --
                       ACCUMULATE           =  <true>              )
```

In this example, error estimation processor ERR6 using the Zienkiewicz-Zhu global smoothing algorithm and Barlow point stress data will be employed for estimating the errors in mesh 1. The ERR6 processor will be run twice, for each element group, followed by the ERRa processor which will accumulate errors by group.

5.3.8 References

None.

5.4 Procedure EST_ERR_SM

5.4.1 General Description

Procedure EST_ERR_SM is a utility procedure for performing finite element solution error estimation involving a stand-alone smoothing processor. It automatically invokes the appropriate smoothing processor followed by an error-estimation post-processor, such as ERRSM, designed to compute errors by comparing raw finite-element stress-type data with smoothed (i.e., nodally continuous) versions of these quantities (see Chapter 9, *Smoothing Processors*, and Chapter 10, *Error Estimation Processors*).

Procedure EST_ERR_SM is typically invoked automatically by solution procedure AR_CONTROL during analyses with adaptive mesh refinement.

5.4.2 Argument Summary

Procedure EST_ERR_SM may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 5.4-1.

Table 5.4-1 Procedure EST_ERR_SM Input Arguments

Argument	Default	Description
CONSTRAINT_SET	1	Specifies constraint-set number for error estimation
ERROR_MEASURE	STRAIN	
ERROR_PROCESSOR	ERRSM	Name of error estimation processor to invoke
GRADIENT_DATASET	GRADS_SM	
GRADIENT_FLAG	<false>	
LDI	1	Logical unit for central database file (*Case*.DBC)
LOAD_SET	1	Specifies load-set number for error estimation
MESH	0	Specifies mesh number for error estimation
NUM_GROUP	0	Number of element groups for error estimation
SAMPLE_LOCATIONS	INTEG_PTS	
SMOOTH_LOCATIONS	ALL	Locations at which smoothed data is to be computed
SMOOTH_OPTIONS	---	Smoothing-processor-specific smoothing options
SMOOTH_PROCESSOR	SMT	Smoothing processor (see Chapter 9)
STEP	0	Specifies load/time-step number for error estimation

5.4.3 Argument Definitions

In this subsection, the procedure arguments summarized in Table 5.4-1 are defined in more detail. The arguments are listed alphabetically and many of the precise definitions are relegated to subordinate procedures and processors, where the actual options are determined. For example, the definition of REFINE_TECHNIQUE depends on which refinement processor the user selects via the REFINE_PROCESSOR argument, and the user is referred to the corresponding refinement processor section in Chapter 11 for details on the options.

5.4.3.1 CONSTRAINT_Argument

This argument defines the constraint set number associated with the element solution data for which error estimates are to be computed. This number should appear as the second cycle number in names of all element solution datasets, e.g., STRESS, STRAIN, and STRAIN_ENERGY (relevant only for linear static analysis).

Argument syntax:

CONSTRAINT_= *constraint_set*

where:

Parameter	Description
constraint_set	Constraint set number (Default value: 1)

5.4.3.2 ERROR_PROCESSOR Argument

This argument defines the error processor to be used for estimating the solution error by comparing smoothed data (to be computed by a stand-alone smoothing processor) with raw finite element data.

Argument syntax:

ERROR_PROCESSOR = *error_processor*

where *error_processor* is the name of the error estimation processor. Only special error estimation processors such as ERRSM can handle pre-smoothed solution data. (Default value: ERRSM)

5.4.3.3 GROUP Argument

This argument defines the element group identity numbers for a group of elements that need to be processed by the ERR*i* processors for each of the element types specified.

Argument syntax:

GROUP = *first:last:incr*
or
GROUP = $g_1,g_2,\ldots,g_N$

where:

Parameter	Description
first	First group ID to be processed (Default value: 0; all groups)
last	Last group ID
incr	Group ID increment
g_i	Group ID

5.4.3.4 LOAD_SET Argument

This argument defines the load set number associated with the element solution data for which error estimates are to be computed. This number appears as the first cycle number in names of all element solution datasets, e.g., STRESS, STRAIN, and STRAIN_ENERGY (relevant only for linear static analysis).

Argument syntax:

LOAD_SET= *load_set*

where:

Parameter	Description
load_set	Load set number. (Default value: 1)

5.4.3.5 LDI Argument

This argument defines the logical device index for the computational database.

Argument syntax:

LDI = *ldi*

where:

Parameter	Description
ldi	Logical device index. (Default value: 1)

5.4.3.6 MESH Argument

This argument defines the mesh number associated with the model and solution data for which error estimates are to be computed. This number should appear as the third cycle number in names of all datasets, e.g., *EltNam*.ERROR.*ldset.conset.mesh*.

Argument syntax:

MESH = *mesh*

where:

Parameter	Description
mesh	Mesh number to be processed. (Default value: 0)

5.4.3.7 SMOOTH_LOCATIONS Argument

This argument defines the locations at which smoothed data is to be computed.

Argument syntax:

SMOOTH_LOCATIONS = *locations*

where:

Parameter	Description
locations	Locations where smoothed data will be computed and stored: INTEG_PTS => element integration points (default) NODES => element nodes BOTH => both integration points and nodes

5.4.3.8 SMOOTH_OPTIONS Argument

This argument defines processor-specific smoothing options.

Argument syntax:

SMOOTH_OPTIONS = *options*

where:

Parameter	Description
options	Smoothing-processor specific option values; see Chapter 9 for details.

5.4.3.9 SMOOTH_PROCESSOR Argument

This argument defines the name of the smoothing processor to run before estimating errors.

Argument syntax:

SMOOTH_PROCESSOR = *processor*

where:

Parameter	Description
processor	Name of a valid smoothing processor. See Chapter 9. (Default: SMZ)

5.4.3.10 STEP Argument

This argument defines the solution step number associated with the element solution data for which error estimates are to be computed. This number appears as the first cycle number in names of all element solution datasets, e.g., STRESS, STRAIN, and STRAIN_ENERGY (relevant only for nonlinear static analysis).

Argument syntax:

STEP = *step*

where:

Parameter	Description
step	Solution step number. (Default value: none)

5.4.4 Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the ERR*i* processor being used. These dataset requirements are documented in detail in Chapter 10.

5.4.5 Current Limitations

EST_ERR_SM is a general purpose procedure and the only limitations on its use are dictated by the limitations of the ERR*i* processor being employed. Refer to individual ERR*i* processors in Chapter 10 for specific processor limitations.

5.4.6 Status and Error Messages

EST_ERR_SM does not print any status or error messages directly. All messages will be produced by the ERR*i* processor being employed. Refer to individual ERR*i* processors in Chapter 10 for specific processor messages.

5.4.7 Examples and Usage Guidelines

5.4.7.1 Example 1: ERROR Estimation Without Group Partition

```
*CALL EST_ERR_SM (      SMOOTHING_PROCESSOR     =  SMZ          ; --
                        ERROR_PROCESSOR         =  ERRSM        ; --
                        ERROR_MEASURE           =  STRAIN       ; --
                        MESH                    =  2              )
```

In this example, error estimation is based on a comparison of the basic finite element strains with a smoothed version of these strains, computed via the Zienkiewicz smoothing processor, SMZ. Error estimation processor ERRSM then computes the element error norms by integrating the strain energy of the difference between the basic strains and smoothed strains over each element domain. The calculations are performed for the finite element solution obtained with mesh 2.

5.4.7.2 Example 2: ERROR Estimation With Group Partition

```
*CALL EST_ERR_SM (      SMOOTHING_PROCESSOR     =  SMZ          ; --
                        ERROR_PROCESSOR         =  ERRSM        ; --
                        ERROR_MEASURE           =  STRAIN       ; --
                        MESH                    =  1            ; --
                        NUM_GROUP               =  2            ; --
                        GROUP                   =  1, 2           )
```

This example is identical to the previous example except that i) error estimation is performed for mesh 1 instead of mesh 2, and ii) smoothing will be performed independently for element groups

1 and 2, which presumably interface with one another at a physical discontinuity such as a non-smooth intersection, a change in material properties, or a concentrated load.

5.4.8 References

None.

5.5 Procedure FACTOR

5.5.1 General Description

Procedure FACTOR is a utility procedure for performing system matrix decomposition. It is automatically invoked by solution procedures such as L_STATIC_1 and NL_STATIC_1 to perform system matrix factorization for a given mesh.

The FACTOR procedure is merely a cover procedure which invokes the appropriate matrix/vector algebra processor to perform the system matrix decomposition task. Existing processors of this type are discussed in Chapter 12, *Matrix/Vector Algebra Processors.*

5.5.2 Argument Summary

Procedure FACTOR may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 5.5-1.

Table 5.5-1 Procedure FACTOR Input Arguments

Argument	Default Value	Description
ASM_MATRIX	1, K	The *ldi* and dataset name of the assembled system matrix
FAC_MATRIX	1, K	The *ldi* and dataset name of the output factored system matrix
FIXED_FRAME	OFF	Fixed-frame option for hierarchical h_s-refinement
LDI_C	1	Logical unit for main COMET-AR database file (*Case*.DBC)
LDI_S	3	Logical unit for system-matrix file (*Case*.DBS)
MATRIX_UPDATE	FULL	Matrix update option for hierarchical h_s-refinement
MESH	0	Mesh number to be analyzed
MTX_BUFFER_SIZE	500000	Matrix buffer size for equation solving
SKY_PROCESSOR	SKY	Linear equation solver processor name
STEP	0	Solution step number

5.5.3 Argument Definitions

In this subsection, the procedure arguments summarized in Table 5.5-1 are defined in more detail. The arguments are listed alphabetically. Refer to Chapter 12, *Matrix/Vector Algebra Processors,* for details on the options.

5.5.3.1 ASM_MATRIX Argument

This argument sets the *ldi* and dataset name of the assembled stiffness matrix.

Argument syntax:

ASM_MATRIX = *ldi, dataset_name*

where *ldi* is the logical device index associated with the system matrix file and *dataset_name* is the assembled system matrix dataset name. (Default value: 1, K)

5.5.3.2 FIXED_FRAME Argument

This argument sets a flag that is relevant only for h_s-refinement. (See Section 12.3 (ASMs) and 12.7 (SKYs) for additional information about this argument).

Argument syntax:

FIXED_FRAME = {<true> | <false>}

Do not change the default setting without the advice of a COMET-AR expert. (Default value: <false>)

5.5.3.3 LDI_C Argument

This argument sets the logical device index associated with the main COMET-AR database file, which must exist before calling FACTOR and must be named *Case*.DBC.

Argument syntax:

LDI_C = *ldi_c*

where *ldi_c* is the logical device index (a positive integer) of the *Case*.DBC file. (Default value: 1)

5.5.3.4 LDI_S Argument

This argument sets the logical device index associated with the system matrix database file, typically named *Case*.DBS.

Argument syntax:

LDI_S = *ldi_s*

where *ldi_s* is the logical device index (a positive integer) of the *Case*.DBS file. If *ldi_s* is not equal to *ldi_c* (see the LDI_C argument) then all system matrices (e.g., stiffness and mass) for the current mesh will be stored on a separate *Case*.DBS file. If *ldi_s* = *ldi_c*, then all system matrices will be stored on the *Case*.DBC file, i.e., a separate *Case*.DBS file will not be created. (Default value: 3)

If a separate *Case*.DBS file is created, it will be deleted and re-created with each new adaptive mesh.

5.5.3.5 MATRIX_UPDATE Argument

This argument sets the matrix-update mode for hierarchical adaptive refinement (relevant only for h_s-refinement).

Argument syntax:

MATRIX_UPDATE = {FULL | PARTIAL}

where FULL implies that the entire stiffness matrix is reformed for each new mesh and thus a complete factorization is required, and where PARTIAL implies that only the updated-mesh contributions to the stiffness matrix are reformed for each new mesh and thus only the new columns added to the assembled matrix require factorization. (Default value: FULL)

5.5.3.6 MESH Argument

This argument sets the number of the mesh to analyze.

Argument syntax:

MESH = *mesh*

where *mesh* is the mesh number. (Default value: 0)

5.5.3.7 MTX_BUFFER_SIZE Argument

This argument sets the size of the memory buffer to be used for matrix factorization by certain matrix/vector algebra processors.

Argument syntax:

MTX_BUFFER_SIZE = *mtx_buffer_size*

where *mtx_buffer_size* is the size of the buffer in logical variables. (Default value: 500000)

5.5.3.8 SKY_PROCESSOR Argument

Selects the matrix/vector algebra processor to be used for factoring the assembled linear equation system.

Argument syntax:

SKY_PROCESSOR = *sky_processor*

where *sky_processor* is the name of the matrix/vector algebra processor. Current options are summarized below.

sky_processor	Description
SKY	Direct solution of skyline matrices by Crout decomposition (LDU) (Default)
SKYs	Direct and/or iterative solution of skyline matrices in conjunction with h_s- and h_t-refinement only
ITER	Iterative solution of compact matrices by PCG algorithm
PVSOLV	Direct solution of skyline matrices optimized for vector computers.

Consult Chapter 12 for more details.

5.5.4 Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the matrix/vector algebra processor being used. These dataset requirements are documented in detail in Chapter 12.

5.5.5 Current Limitations

FACTOR is a general purpose procedure and the only limitations on its use are dictated by the limitations of the equation solver processor being employed. Refer to individual matrix/vector algebra processors in Chapter 12 for specific processor limitations.

5.5.6 Status and Error Messages

FACTOR does not print any status or error messages directly. All messages will be produces by the equation solver processor being employed. Refer to individual matrix/vector algebra processor in Chapter 12 for specific processor messages.

5.5.7 Examples and Usage Guidelines

5.5.7.1 Example 1: In-Core Factorization

```
*CALL FACTOR (     SKY_PROCESSOR      =  SKY                          ; --
                   ASM_MATRIX         =  3, SYSTEM.MATRIX...2         ; --
                   FAC_MATRIX         =  3, SYSTEM.MATRIX...2         ; --
                   MESH               =  2                             )
```

In this example, the SKY processor will be used to factor in-core an assembled skyline matrix existing on *ldi 3*, in the dataset SYSTEM.MATRIX...2. The factored matrix will overwrite the assembled matrix since the same dataset name is specified for both matrices.

5.5.7.2 Example 2: Out-of-Core Factorization

```
*CALL FACTOR (     SKY_PROCESSOR      =  SKYS                         ; --
                   MTX_BUFFER_SIZE    =  100000                       ; --
                   MATRIX_UPDATE      =  FULL                         ; --
                   MESH               =  2                             )
```

In this example, the SKYs processor will be used to fully factor out-of-core the assembled skyline matrix of mesh 2. The factorization will be performed out-of-core using only 100000 words of physical memory.

5.5.8 References

None.

5.6 Procedure FORCE

5.6.1 General Description

This section describes the FORCE Utility Procedure, which directs the generation of nodal force vectors (internal, external, or residual). The main purpose of this procedure is to invoke the appropriate element processors for adding element load contributions to nodal force vectors.

5.6.2 Argument Summary

Procedure FORCE may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 5.6-1.

Table 5.6-1 Procedure FORCE Input Arguments

Argument	Default Value	Description
COROTATION	<false>	Sets the default element corotational option
DISPLACEMENT	—	Sets the default name of nodal displacement dataset
INPUT_FORCE	DUMMY.FORCE	Sets the default name of nodal external force dataset
LDI	1	Sets the default *ldi* of computational database library
LOAD_FACTOR	1.0	Sets the default load factor to be applied to element loads
LOAD_SET	1	Sets the default load set number for element loads
MESH	0	Sets the mesh number
NL_GEOM	<false>	Sets the default geometric nonlinearity option
NL_LOAD	<false>	Sets the default load nonlinearity option
OUTPUT_FORCE	SYS.FORCE	Sets the default *ldi* and dataset name of output force vector
ROTATION	---	Sets the default name of nodal rotation pseudovector dataset
SE_TOT	<false>	
STEP	0	
TYPE	RESIDUAL	Sets the type of force to be computed

5.6.3 Argument Definitions

In this subsection, the procedure arguments summarized in Table 5.6-1 are defined in more detail. The arguments are listed alphabetically. Refer to Chapter 7, *Element Processors*, for more details on the options.

5.6.3.1 COROTATION Argument

This argument sets the default element corotational option for geometrically nonlinear analysis. The corotational capability is built in to the generic element processor (ES) and enables beam and shell elements to be employed with arbitrarily large rotations (but small to moderate strains) even if the element strain-displacement relations do not intrinsically account for large rotations exactly.

Argument syntax:

COROTATION = *corotation_option*

where

corotation_option	Description
0 or <false>	Element corotation will not be used. (Default)
1	Basic element corotation will be used. This option is sufficient unless True-Newton iteration is begin performed at the nonlinear solution procedure level.
2	Higher-order element corotation will be used. This option should be used only if True-Newton iteration has been selected at the nonlinear solution procedure level; and even then may provide only marginal improvement in nonlinear convergence over option 1. It adds additional terms to the tangent stiffness matrix that render it more consistent.

5.6.3.2 DISPLACEMENT Argument

This argument changes the default *ldi* and name of the nodal displacement dataset.

Argument syntax:

DISPLACEMENT = *ds_name*

where *ds_name* is the nodal displacement dataset name. (Default value: None)

5.6.3.3 INPUT_FORCE Argument

This argument changes the default name of the nodal force dataset.

Argument syntax:

INPUT_FORCE = *ds_name*

where *ds_name* is the dataset name. (Default value: None)

5.6.3.4 OUTPUT_FORCE Argument

This argument changes the default name of the computed (output) force vector dataset.

Argument syntax:

OUTPUT_FORCE = *ds_name*

where *ds_name* is the new dataset name. (Default value: 1, SYS.FORCE)

5.6.3.5 LOAD_FACTOR Argument

This argument changes the default load factor to be applied to all element loads.

Argument syntax:

LOAD_FACTOR = *load_factor*

where *load_factor* is a floating-point scale factor. (Default value: 1.0)

5.6.3.6 LOAD_SET Argument

This argument changes the default load set number for element loads during either load definition or consistent external force formation.

Argument syntax:

LOAD_SET = *load_set*

where *load_set* is an integer load-set number. (Default value: 1)

5.6.3.7 LDI Argument

This argument changes the default logical device index (*ldi*) for all datasets input/output by the current ES*i* processor, except those for which an explicit *ldi* is used in a separate database command (e.g., OUTPUT_FORCE).

Argument syntax:

LDI = *ldi*

where *ldi* is the logical device index of the database library. (Default value: 1)

5.6.3.8 MESH Argument

This argument changes the default mesh number used in all dataset names (unless otherwise specified via a separate dataset command).

Argument syntax:

MESH = *mesh*

where *mesh* is an integer number, typically set to the current mesh number. (Default value: 0)

5.6.3.9 NL_GEOM Argument

This argument changes the default geometric nonlinearity option. It is often used in conjunction with the COROTATION command.

Argument syntax:

NL_GEOM = *nl_geom_option*

where:

nl_geom_option	Description
0 or <false>	The analysis is geometrically linear; linear element strain-displacement relations will be employed, and element corotational will be disregarded. (Default)
1	The analysis is geometrically nonlinear, but only linear element strain-displacement relations will be used. With this option, geometric nonlinearity must be accounted for via element corotation (see COROTATION command), which for many beam/shell element types is not as accurate as option 2.
2	The analysis is geometrically nonlinear, and nonlinear element strain-displacement relations will be used. Element corotation may or not be selected with this option. For many beam/shell element types, nonlinear element strain-displacement relations enhances corotation, making it more accurate for a given mesh and rotation magnitude.

5.6.3.10 NL_LOAD Argument

This argument changes the default load nonlinearity option. It affects whether "live" loads are to be processed as part of the external force vector, or the tangent stiffness matrix.

Argument syntax:

NL_LOAD = *nl_load_option*

where

nl_load_option	Description
0 or <false>	Ignore load nonlinearity (i.e., displacement dependence). Only displacement-independent ("dead") external loads are to be processed in the following FORM FORCE or FORM STIFFNESS command. (Default)
1	Include load nonlinearity. Only displacement-dependent ("live") external loads are to be processed in the following FORM FORCE or FORM STIFFNESS command.

5.6.3.11 ROTATION Argument

This argument changes the default name of the nodal rotation (pseudovector) dataset.

Argument syntax:

ROTATION = *ds_name*

where *ds_name* is the new dataset name. (Default value: NODAL.ROTATION.1.1)

5.6.3.12 TYPE Argument

This argument defines the type of force to be computed.

Argument syntax:

TYPE = *force_type*

where *force_type* of force to be computed INTERNAL, EXTERNAL, or RESIDUAL. (Default value: RESIDUAL)

5.6.4 Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the ES*i* processor being used. These dataset requirements are documented in detail in Chapter 7.

5.6.5 Current Limitations

FORCE is a general purpose procedure and the only limitations on its use are dictated by the limitations of the ES*i* processor being employed. Refer to individual ES*i* processors in Chapter 7 for specific processor limitations.

5.6.6 Status and Error Messages

FORCE does not print any status or error messages directly. All messages will be produced by the ES*i* processor being employed. Refer to individual ES*i* processors in Chapter 7 for specific processor messages.

5.6.7 Examples and Usage Guidelines

5.6.7.1 Example 1: External Load Vector

```
*call FORCE (     TYPE              =  EXTERNAL                    ; --
                  INPUT_FORCE       =  1, NODAL.SPEC_FORCE.1..2    ; --
                  OUTPUT_FORCE      =  1, NODAL.EXT_FORCE.1..2     ; --
                  NL_GEOM           =  <false>                     ; --
                  COROTATION        =  <false>                     ; --
                  NL_LOAD           =  <false>                     ; --
                  MESH              =  2                           ; --
                  LOAD_SET          =  1                             )
```

In this example, the element loads will be added to the nodal applied forces and the resulting nodal load vector will be stored in a dataset named NODAL.EXT_FORCE.1..2 in the file associated with logical device index 1.

5.6.8 References

None.

5.7 Procedure INITIALIZE

5.7.1 General Description

Procedure INITIALIZE is a utility procedure for performing solution initialization tasks. It is automatically invoked by solution procedures such as L_STATIC_1 and NL_STATIC_1 to perform initialization for a given finite element mesh.

Procedure INITIALIZE performs a sequence of calls to other procedures and processors as shown in Figure 5.7-1.

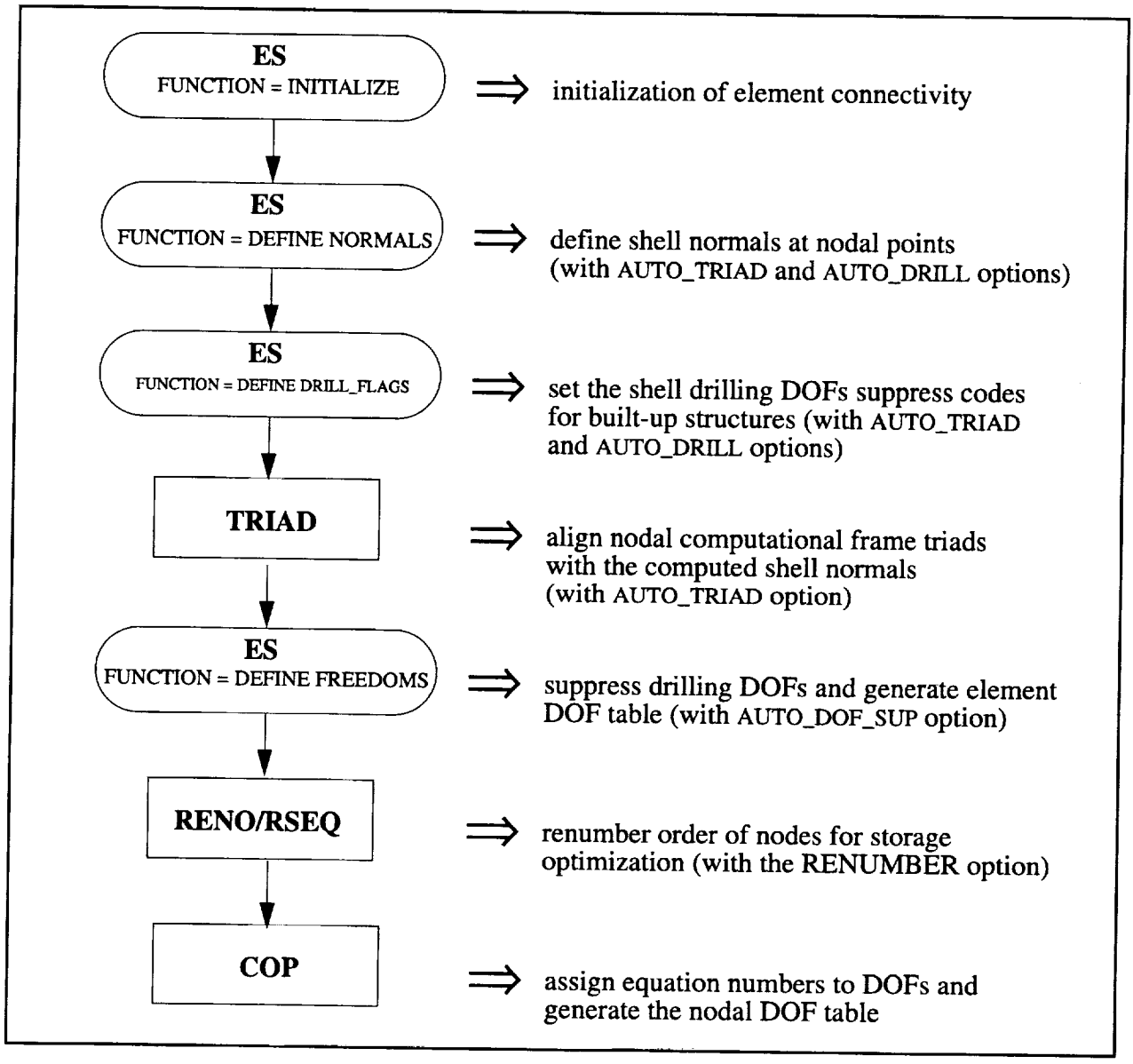


Figure 5.7-1 INITIALIZE: Model Initialization Steps

The INITIALIZE procedure is merely a cover procedure invoking a sequence of utility procedures and processors to perform the solution initialization task for a given model/mesh. Each of these other utility procedures is described in the current chapter; the processors are described in Part III.

5.7.2 Argument Summary

Procedure INITIALIZE may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 5.7-1.

Table 5.7-1 Procedure INITIALIZE Input Arguments

Argument	Default Value	Description
AUTO_DOF_SUP	<false>	Automatic DOF suppression switch
AUTO_DRILL	<false>	Automatic drilling stiffness augmentation switch
AUTO_MPC	<false>	
AUTO_TRIAD	<false>	Automatic triad re-alignment for drilling DOFs
CONSTRAINT_SET	1	Constraint set number to be used for suppressing DOFs in the assembled system matrix prior to factorization
LDI	1	Logical unit for main COMET-AR database file (*Case*.DBC)
MATRIX_UPDATE	FULL	Matrix update option for hierarchical h_s-refinement
MESH	0	Mesh number to be analyzed
REFINE_TECHNIQUE	ht	Mesh refinement technique (h_t => transition h)
RENO_PROCESSOR	RSEQ	Node renumbering processor
RENUMBER_OPT	3	Node renumbering option

5.7.3 Argument Definitions

In this subsection, the procedure arguments summarized in Table 5.7-1 are defined in more detail. The arguments are listed alphabetically.

5.7.3.1 AUTO_DOF_SUP Argument

Automatic DOF (degree-of-freedom) suppression switch. This capability automatically suppresses extraneous DOFs, especially useful during adaptive mesh refinement. It is described in more detail in Section 2.10, *Automatic DOF Suppression and Drilling Stabilization.*

Argument syntax:

AUTO_DOF_SUP = *option* [, *angle_tol*]

where

Parameter	Description
option	Automatic DOF suppression option switch: {<true> | <false>}. If <true>, all DOFs (in the computational frame) that are unsupported by element stiffness will be suppressed throughout the adaptive refinement process. (Default value: <true>)
angle_tol	Angle tolerance to use for suppression of shell element drilling DOFs; see Section 2.10 for details. (Default value: depends on element type)

In most cases, it is best to leave the default setting intact.

5.7.3.2 AUTO_DRILL Argument

Automatic drilling stiffness option. This option causes shell elements to add artificial drilling rotational stiffness to nodal DOFs that would otherwise be unstable computationally. See Section 2.10 and individual element processor sections in Chapter 7 for more information.

Argument syntax:

AUTO_DRILL = *option* [, *angle_tol*, *scale_fac*]

where

Parameter	Description
option	Automatic drilling stiffness switch: {<true> | <false>}. If <true>, certain shell element types will add artificial drilling stiffness to nodal DOFs that require stabilization. (Default value: <false>)
angle_tol	Angle tolerance to use for determining whether artificial drilling stiffness is needed at a given node. See Chapter 2 for details. (Default value: depends on element type)
scale_fac	Scale factor determining magnitude of artificial drilling stiffness to be added by selected shell elements. See Chapter 2 for interpretation. (Default value: depends on element type)

AUTO_DRILL is not recommended for nonlinear analysis.

5.7.3.3 AUTO_TRIAD Argument

Automatic computational triad (i.e., DOF direction) re-alignment option. This option is an alternative to AUTO_DRILL that causes re-alignment of the computational triads at all nodes that require drilling DOF stabilization as long as no boundary conditions have been defined there. The

computational axes are re-aligned such that one of them is parallel to the average element surface-normal at the node. Then, extraneous (unstable) drilling rotational DOFs can be subsequently suppressed via the AUTO_DOF_SUP option. (See Section 2.10, *Automatic DOF Suppression and Drilling Stabilization*, for more information.)

Argument syntax:

AUTO_TRIAD = *option* [, *angle_tol*]

where

Parameter	Description
option	Automatic triad re-alignment option switch: {<true> | <false>}. If <true>, computational triads will be re-aligned with the average element normal at all nodes that require drilling DOF stabilization, unless boundary conditions are defined there. (Default value: <false>)
angle_tol	Angle tolerance to use for determining whether drilling stabilization is needed at a given node. See Section 2.10 for details. (Default value: depends on element type)

AUTO_TRIAD should only be used in conjunction with AUTO_DOF_SUP and cannot be used in conjunction with user-defined point forces and/or multi-point constraints.

5.7.3.4 CONSTRAINT_SET Argument

This argument defines the constraint set number associated with the element solution data for which error estimates are to be computed. This number should appear as the second cycle number in names of all element solution datasets, e.g., STRESS, STRAIN, and STRAIN_ENERGY (relevant only for linear static analysis).

Argument syntax:

CONSTRAINT_SET = *constraint_set*

where

Parameter	Description
constraint_set	Constraint set number (Default value: 1)

5.7.3.5 LDI Argument

This argument sets the logical device index associated with the main COMET-AR database file, which must exist before calling INITIALIZE and must be named *Case*.DBC.

Argument syntax:

LDI = *ldi*

where *ldi* is the logical device index (a positive integer) of the *Case*.DBC file. (Default value: 1)

5.7.3.6 MATRIX_UPDATE Argument

This argument sets the matrix-update mode for hierarchical adaptive refinement (relevant only for h_s-refinement).

Argument syntax:

MATRIX_UPDATE = {FULL | PARTIAL}

where FULL implies that the entire stiffness matrix is reformed for each new mesh, and where PARTIAL implies that only the updated-mesh contributions to the stiffness matrix are reformed for each new mesh. (Default value: FULL)

5.7.3.7 MESH Argument

This argument sets the number of the mesh to analyze.

Argument syntax:

MESH = *mesh*

where *mesh* is the mesh number. (Default value: 0)

5.7.3.8 REFINE_TECHNIQUE Argument

This argument sets the refinement technique to be employed by the mesh refinement processor (REF*i*) specified via the REFINE_PROCESSOR argument.

Argument syntax:

REFINE_TECHNIQUE = *refine_technique*

where *refine_technique* is the name of the refinement technique. For example, in conjunction with processor REF1, the REFINE_TECHNIQUE argument might be set equal to h_r, h_c, h_s, or p (among others). See the documentation under specific REF*i* processors in Chapter 11 for details. (Default value: h_t)

5.7.3.9 RENO_PROCESSOR Argument

This argument sets the name of the equation (or node) renumbering processor to be used to optimize matrix equation solving (time and/or storage).

Argument syntax:

RENO_PROCESSOR = *renumber_processor*

where *renumber_processor* is the processor name. Current options are summarized below.

renumber_processor	Description
RENO	Node renumbering using a geometric algorithm (Default)
RSEQ	Node renumbering via a variety of order-optimization algorithms

Consult the appropriate section in Chapter 6, *Pre-Processors* for more details.

5.7.3.10 RENUMBER Argument

Sets a flag determining whether or not to perform equation renumbering (e.g., bandwidth, skyline or sparsity optimization) both initially and whenever the mesh is updated by adaptive refinement.

Argument syntax:

RENUMBER = *renumber_flag*

where *renumber_flag* may be set either to <true> or <false>. (Default value: <true>)

5.7.3.11 RENUMBER_OPT

This argument sets the equation renumbering option to use within the renumbering processor selected via the RENO_PROCESSOR argument (assuming RENUMBER = <true>).

Argument syntax:

RENUMBER_OPT = *renumber_option*

where *renumber_option* indicates the renumbering option and depends on the particular renumbering processor chosen. See processors RENO, RSEQ, etc., in Chapter 6. (Default value: 0)

5.7.4 Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the ES*i* processor being employed, and by the renumbering and REDO processors. These dataset requirements are documented in detail in Chapter 6, *Pre-Processors*, and Chapter 7, *Element Processors*.

5.7.5 Current Limitations

INITIALIZE is a general purpose procedure and the only limitations on its use are dictated by the limitations of the ES*i* processor being employed, renumbering, and the REDO processors. Refer to Chapters 6 and 7 for specific processor limitations.

5.7.6 Status and Error Messages

INITIALIZE does not print any status or error messages directly. All messages will be produced by the ES*i* processor being employed and by the renumbering and the REDO processors. Refer to Chapters 6 and 7 for specific processor messages.

5.7.7 Examples and Usage Guidelines

5.7.7.1 Example 1: Initialization with Auto DOF Suppression

```
*call INITIALIZE (      AUTO_DOF_SUP            =  <true>     ; --
                        AUTO_DRILL              =  <false>    ; --
                        AUTO_TRIAD              =  <false>    ; --
                        RENUMBER                =  <true>     ; --
                        RENO_PROCESSOR          =  RSEQ       ; --
                        RENO_OPTION             =  0          ; --
                        MESH                    =  2          ; --
                        REFINEMENT_TECHNIQUE    =  hc         ; --
                        LDI                     =  1          ; --
                        CONSTRAINT_SET          =  1            )
```

In this example, mesh 2 model is initialized using the automatic DOF suppression option. The nodal points will be reordered using RSEQ processor and renumbering method 0.

5.7.8 References

None.

5.8 Procedure REF_MESH_1

5.8.1 General Description

Procedure REF_MESH_1 is a utility procedure for performing one pass of adaptive mesh refinement based on a single solution and corresponding error estimates. This procedure is a cover that invokes adaptive mesh refinement processors such as REF1, described in Chapter 11. It is typically called via procedure AR_CONTROL.

5.8.2 Argument Summary

Procedure REF_MESH_1 may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 5.8-1.

Table 5.8-1 Procedure REF_MESH_1 Input Arguments

Argument	Default Value	Description
CONSTRAINT_SET	1	Specifies the constraint set number
CONVERGE_TOL	0.05	Global error tolerance (relative error)
H_GRADIENT	0.8	Relative energy gradient mark above which both h and p-refinement will occur (for mixed *h*/*p*-refinement options)
LDI	1	Logical unit for main COMET-AR database file (*Case*.DBC)
LDI_GM	7	
LOAD_SET	1	Specifies the load-set number
MAX_ASPECT_RATIO	2.0, 2.0	Distortion control parameters for ht refinement
MAX_H_LEVEL	10	Maximum levels of *h*-refinement for any element
MAX_P_LEVEL	0	Maximum levels of *p*-refinement globally
NEW_MESH	0	The refined mesh number
NUM_REFINE_TOLS	1	No. of error tolerances guiding refinement
NUM_UNREFINE_TOLS	0	
OLD_MESH	0	Mesh from which to restart.
P_GRADIENT	0.0	Relative energy gradient mark below which only *p*-refinement will occur (for mixed *h*/*p*-refinement options)
REFINE_DIRS	1, 2	Refinement directions (1,2—implies 2D)
REFINE_INDICATOR	MAX_RATIO	Type of refinement indicator
REFINE_LEVELS	1	List of refinement levels corresponding to REFINE_TOLS
REFINE_PROCESSOR	REF1	Name of mesh refinement processor
REFINE_TECHNIQUE	ht	Mesh refinement technique (h_t => transition h)

Table 5.8-1 Procedure REF_MESH_1 Input Arguments

Argument	Default Value	Description
REFINE_TOLS	0.05	List of local (element) error tolerances for refinement
STEP	0	Specifies the solution step number
UNREFINE_LEVELS	0	
UNREFINE_TOLS	.00	

5.8.3 Argument Definitions

In this subsection, the procedure arguments summarized Table 5.8-1 are defined in more detail. The arguments are listed alphabetically. Refer to the corresponding refinement processor section in Part III for details on the options.

5.8.3.1 CONSTRAINT_SET Argument

This argument defines the constraint set number associated with the element and nodal data in both the reference and the refined meshes. This number should appear as the second cycle number in names of all element and nodal datasets.

Argument syntax:

CONSTRAINT_SET = *conset*

where *conset* is the constraint set number (Default value: 1)

5.8.3.2 CONVERGE_TOL Argument

This argument sets the value of the adaptive mesh refinement (AR) global convergence tolerance. This is a relative error measure (in fractional form) below which convergence of the discrete solution to the governing equations is assumed and no further adaptive mesh refinement is performed. The quantitative interpretation of this error measure depends on the particular error estimation processor (ERR*i*) and refinement processor (REF1) selected by the user (see ERROR_PROCESSOR and REF_PROCESSOR arguments).

Argument syntax:

CONVERGE_TOL = *converge_tol*

where *converge_tol* is the relative error tolerance in fractional form (e.g., .1 corresponds to 10 percent error). (Default value: .05)

5.8.3.3 H_GRADIENT Argument

This argument defines the *h_gradient* mark on the element energy gradient axis for multi-technique refinement (see the "REF1—Multi-Level and Multi-Technique Refinement Control" subsection for details).

Argument syntax:

H_GRADIENT = *h_gradient*

where *h_gradient* is the h_gradient mark value. (Default value: 0.8)

5.8.3.4 LDI Argument

This argument sets the logical device index associated with the main COMET-AR database file, which must exist before calling REF_MESH_1, and be named *Case*.DBC.

Argument syntax:

LDI = *ldi*

where *ldi* is the logical device index (a positive integer) of the *Case*.DBC file. (Default value: 1)

5.8.3.5 LOAD_SET Argument

This argument defines the load set number associated with the element and nodal data in both the reference and the refined meshes. This number should appear as the first cycle number in names of all element and nodal datasets.

Argument syntax:

LOAD_SET = *ldset*

where *ldset* is the load set number (Default value: 1)

5.8.3.6 MAX_ASPECT_RATIO Argument

Sets the maximum element aspect ratios before and after prospective adaptive mesh refinement.

Argument syntax:

MAX_ASPECT_RATIO = *before, after*

where *before* denotes the maximum element aspect ratio before a prospective mesh refinement, and *after* denotes the maximum element aspect ratio after a prospective mesh refinement. If either of these limits would be violated, an alternate element refinement pattern is selected. This argument is relevant primarily for transition-based (h_t) refinement, where aspect ratios can be used to control the degree of element distortion. See Chapter 11 for more information. (Default value: 2.0,2.0)

5.8.3.7 MAX_H_LEVEL Argument

Sets the maximum number of levels of adaptive *h*-refinement allowed within any one element. If the mesh refinement processor (REF*i*) determines that more than this many levels of *h*-refinement are necessary to achieve convergence, the adaptive analysis is terminated.

Argument syntax:

MAX_H_LEVEL = *max_h_level*

where *max_h_level* denotes the maximum number of levels of *h*-refinement permitted by the user for any one element. See Chapter 11 for more information. (Default value: 10)

5.8.3.8 MAX_P_LEVEL Argument

Sets the maximum number of levels of uniform *p*-refinement allowed for the model. If the mesh refinement processor (REF*i*) determines that more than this many levels of *p*-refinement are necessary to achieve convergence, the adaptive analysis is terminated.

Argument syntax:

MAX_P_LEVEL = *max_p_level*

where *max_p_level* denotes the maximum number of levels of uniform *p*-refinement permitted. See Chapter 11 for more information. (Default value: 0)

5.8.3.9 NEW_MESH Argument

This argument sets the mesh number of the refined (output) mesh.

Argument syntax:

NEW_MESH = *new_mesh*

where *new_mesh* is the mesh number of the refined mesh. (Default value: 0)

5.8.3.10 NUM_REFINE_TOLS Argument

Sets the number of local (element) error tolerances that will be used to guide adaptive refinement. The REFINE_TOLS argument specifies the error values for these tolerances, and the REFINE_LEVELS argument indicates the number of levels of refinement to perform when each tolerance is exceeded.

Argument syntax:

NUM_REFINE_TOLS = *num_refine_tols*

where *num_refine_tols* denotes the number of refinement tolerances. See Chapter 11 for more information. (Default value: 1)

5.8.3.11 OLD_MESH Argument

Sets the number of the mesh to be refined.

Argument syntax:

OLD_MESH = *old_mesh*

where *old_mesh* denotes the mesh number of the mesh to be refined. (Default value: 0)

5.8.3.12 P_GRADIENT Argument

This argument defines the P_gradient mark on the element energy gradient axis for multi-method refinement (see the "REF1—Multi-Level and Multi-Technique Refinement Control" subsection for details).

Argument syntax:

P_GRADIENT = *p_gradient*

where *p_gradient* is the p_gradient mark value. (Default value: 0.0)

5.8.3.13 REFINE_DIRS Argument

Establishes a list of intrinsic element directions in which to allow adaptive refinement.

Argument syntax:

REFINE_DIRS = *dir1* [, *dir2* [, *dir3*]]

where *dir1*, *dir2,* and *dir3* are intrinsic element direction numbers (i.e., in the elements internal, or natural, coordinate system), and each may take on a value between 1 and the maximum number of intrinsic element dimensions (i.e., 3 for 3D elements, 2 for 2D elements, and 1 for 1D elements). This can eliminate unnecessary refinement in, for example, axisymmetric shell problems, where only one of the surface directions need be refined. See Chapter 11 for more information. (Default value: 1, 2).

5.8.3.14 REFINE_INDICATOR Argument

Sets the type of element refinement indicator to be used by the adaptive refinement processor (see Chapter 11). The refinement indicator is the criterion used to determine whether an element's error estimate is high enough to warrant refinement. The values of the refinement indicator denoting various levels of refinement are set by the REFINE_TOLERANCES argument.

Argument syntax:

REFINE_INDICATOR = *refine_indicator*

where *refine_indicator* denotes the name of the element refinement indicator to be used. (Default value: AVE; see Chapter 11 for details.)

5.8.3.15 REFINE_LEVELS Argument

Sets an array of element refinement levels corresponding to the array of refinement tolerances specified via the REFINE_TOLS argument. An element refinement level is defined as one application of local refinement, employing the refinement type specified via the REFINE_TECHNIQUE argument (e.g., h_p h_c, h_s or p).

Argument syntax:

REFINE_LEVELS = *ref_lev_1, ref_lev_2, ... ref_lev_NUM_REFINE_TOLS*

where *ref_lev_"i"* denotes the number of levels to refine an element when the element refinement (error) indicator exceeds the tolerance specified by *ref_tol_"i"* in the REFINE_TOLS argument; and NUM_REFINE_TOLS is the value set in the NUM_REFINE_TOLS argument (see Chapter 11 for details). (Default value: 1)

5.8.3.16 REFINE_PROCESSOR Argument

Sets the name of the mesh refinement processor (REF*i*) to be invoked by the REF_MESH_1 procedure.

Argument syntax:

REFINE_PROCESSOR = *refine_processor*

where *refine_processor* is the name of the mesh refinement processor. Current options are summarized below.

refine_processor	Description
REF1	Contains a variety of adaptive mesh refinement techniques (Default)

Consult Chapter 11 for more details.

5.8.3.17 REFINE_TECHNIQUE Argument

Sets the refinement technique to be employed by the mesh refinement processor (REF*i*) specified via the REFINE_PROCESSOR argument.

Argument syntax:

REFINE_TECHNIQUE = *refine_technique*

where *refine_technique* is the name of the refinement technique. For example, in conjunction with processor REF1, the REFINE_TECHNIQUE argument might be set equal to *ht*, *hc*, or *p* (among others). See Chapter 11 for details. (Default value: ht)

5.8.3.18 REFINE_TOLS Argument

Sets an array of element refinement tolerances corresponding to the array of refinement levels specified via the REFINE_LEVELS argument. An element refinement tolerance is a limit in the value of the element error-based refinement indicator (see the REFINE_INDICATOR argument) beyond which an element is refined by a prescribed number of levels.

Argument syntax:

REFINE_TOLS = *ref_tol_1, ref_tol_2, ... ref_tol_NUM_REFINE_TOLS*

where *ref_tol_"i"* denotes the value of the element refinement indicator beyond which an element should be refined by *ref_lev_"i"* levels, where *ref_lev_"i"* is specified in the REFINE_LEVELS argument; and NUM_REFINE_TOLS is the value set in the NUM_REFINE_TOLS argument (see Chapter 11 for details). (Default value: .05)

5.8.4 Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the REF*i* processor being used. These dataset requirements are documented in detail in Chapter 11.

5.8.5 Current Limitations

REF_MESH_1 is a general purpose procedure and the only limitations on its use are dictated by the limitations of the REF*i* processor being employed. Refer to individual REF*i* processors in Chapter 11 for specific processor limitations.

5.8.6 Status and Error Messages

REF_MESH_1 does not print any status or error messages directly. All messages will be produced by the REF*i* processor being employed. Refer to individual REF*i* processors in Chapter 11 for specific processor messages.

5.8.7 Examples and Usage Guidelines

5.8.7.1 Example 1: Constraint-Based Refinement (h_c)

```
*call REF_MESH_1 (      REFINE_PROCESSOR      =  REF1                ; --
                        REFINE_TECHNIQUE      =  hc                  ; --
                        REFINE_INDICATOR      =  AVE                 ; --
                        NUM_REFINE_TOLS       =  1                   ; --
                        REFINE_TOLS           =  0.05                ; --
                        REFINE_LEVELS         =  1                   ; --
                        OLD_MESH              =  0                   ; --
                        NEW_MESH              =  1                   ; --
                        FUNCTION              =  FORM STIFFNESS       )
```

In this example, reference mesh 0 is being refined (the refined mesh will be mesh 1) by up to one level of refinement using constraint-based refinement technique (h_c-refinement). Each element for which the relative element error is greater than 5% will be refined by dividing it into four elements.

5.8.8 References

None.

5.9 Procedure SOLVE

5.9.1 General Description

Procedure SOLVE is a utility procedure for solving a system of linear equations. It is automatically invoked by solution procedures such as L_STATIC_1 and NL_STATIC_1 to compute the system displacement vector solution for a given finite element mesh.

Procedure SOLVE performs a sequence of calls to utility procedures and matrix/vector algebra processors to complete the following solution steps:

- Assemble the system load vector using an assembly processor.
- Solve the system of equations to obtain the solution vector using an equation-solver processor.
- Construct the nodal solution vector from the system vector using the COP processor.

The SOLVE procedure is merely a simple cover procedure invoking a sequence of utility processors to perform the solution tasks. These utility processors are discussed in Chapter 12, *Matrix/Vector Processors*.

5.9.2 Argument Summary

Procedure SOLVE may be invoked with the COMET-AR *CALL directive employing the arguments summarized in Table 5.9-1.

Table 5.9-1 Procedure SOLVE Input Arguments

Argument	Default Value	Description
ASM_PROCESSOR	ASM	Matrix/vector assembly processor
CONSTRAINT_SET	1	Constraint set number to be used for suppressing DOFs in the assembled system matrix prior to factorization
ELT_MATRIX	—	Logical unit and dataset name for the element stiffness matrices
FIXED_FRAME	OFF	Fixed-frame option for hierarchical h_s-refinement
LDI_C	1	Logical unit for main COMET-AR database file (*Case*.DBC)
LDI_E	2	Logical unit for element-matrix file (*Case*.DBE)
LDI_S	3	Logical unit for system-matrix file (*Case*.DBS)
LOAD_FACTOR	1.0	Load factor to be applied to the right hand side load vector prior to the solution
LOAD_SET	1	Load set number to be used as the external force vector

Table 5.9-1 Procedure SOLVE Input Arguments (Continued)

Argument	Default Value	Description
MATRIX	—	Logical unit and dataset name for the assembled and factored system matrix
MATRIX_UPDATE	FULL	Matrix update option for hierarchical h_s-refinement
MAX_ITER	100	Maximum iterations for iterative solvers
MESH	0	Mesh number to be analyzed
MTX_BUFFER_SIZE	500000	Matrix buffer size for equation solving
REACTION		Compute reactions at specified boundary points
REFINE_TECHNIQUE	ht	Mesh refinement technique (h_t => transition h)
RHS	—	Logical unit and dataset name for the right hand side load vector
SKY_PROCESSOR	SKY	Linear equation solver processor name
SOLN	—	Logical unit and dataset name for the solution vector
SPEC_DISP	—	Logical unit and dataset name for the nodal specified displacement table
SOLVER_CONV_TOL	0.000001	Convergence tolerance for iterative solvers
STEP	0	Solution step number

5.9.3 Argument Definitions

In this subsection, the procedure arguments summarized Table 5.9-1 are defined in more detail. The arguments are listed alphabetically. See Chapter 12, *Matrix/Vector Processors* for detailed description of the options.

5.9.3.1 ASM_PROCESSOR Argument

Selects the matrix assembly processor to be used for assembling element (stiffness/mass) matrices into corresponding system matrices.

Argument syntax:

ASM_PROCESSOR = *asm_processor*

where *asm_processor* is the name of the matrix assembly processor. Current options include ASM (for h_t and h_c types of mesh refinement) and ASMs (h_s mesh refinement). (Default value: ASM)

5.9.3.2 CONSTRAINT_SET Argument

This argument defines the constraint set number associated with the element, nodal, and system data. This number should appear as the second cycle number in names of all datasets.

Argument syntax:

CONSTRAINT_SET = *conset*

where:

Parameter	Description
conset	Constraint set number (Default value: 1)

5.9.3.3 ELT_MATRIX Argument

This argument sets the logical device index and dataset name for the element matrices (stiffness).

Argument syntax:

ELT_MATRIX = *ldi, dataset_name*

where *ldi* is the logical device index for the file containing the matrices, and *dataset_name* is the name of the element matrices dataset. (Default value: None)

5.9.3.4 FIXED_FRAME Argument

Sets a flag that is relevant only for h_s-refinement.

Argument syntax:

FIXED_FRAME = {<true> | <false>}

Do not change the default setting without the advice of a COMET-AR expert. (Default value: <false>)

5.9.3.5 REACTION Argument

This argument sets the reaction force computation switch.

Argument syntax:

REACTION = *flag*

where *flag* is the switch option. (Default value: <false>—do not compute reaction forces)

5.9.3.6 LDI_C Argument

This argument sets the logical device index associated with the main COMET-AR database file, which must exist before calling SOLVE and must be named *Case*.DBC.

Argument syntax:

LDI_C = *ldi_c*

where *ldi_c* is the logical device index (a positive integer) of the *Case*.DBC file. (Default value: 1)

5.9.3.7 LDI_E Argument

This argument sets the logical device index associated with the element matrix database file, typically named *Case*.DBE.

Argument syntax:

LDI_E = *ldi_e*

where *ldi_e* is the logical device index (a positive integer) of the *Case*.DBE file. If *ldi_e* is not equal to *ldi_c* (see the LDI_C argument) then all element matrices (e.g., stiffness and mass) for the current mesh will be stored on a separate *Case*.DBE file; however, if *ldi_e* = *ldi_c*, then all element matrices will be stored on the *Case*.DBC file; i.e., a separate *Case*.DBE file will not be created. (Default value: 2)

If a separate *Case*.DBE file is created, it will be deleted and re-created with each new adaptive mesh.

5.9.3.8 LDI_S Argument

This argument sets the logical device index associated with the system matrix database file, typically named *Case*.DBS.

Argument syntax:

LDI_S = *ldi_s*

where *ldi_s* is the logical device index (a positive integer) of the *Case*.DBS file. If *ldi_s* is not equal to *ldi_c* (see the LDI_C argument) then all system matrices (e.g., stiffness and mass) for the current mesh will be stored on a separate *Case*.DBS file; however, if *ldi_s* = *ldi_c*, then all system matrices will be stored on the *Case*.DBC file, i.e., a separate *Case*.DBS file will not be created. (Default value: 3)

If a separate *Case*.DBS file is created, it will be deleted and re-created with each new adaptive mesh.

5.9.3.9 LOAD_FACTOR Argument

This argument sets the value for the load factor to be applied to the load vector prior to solution.

Argument syntax:

LOAD_FACTOR = *factor*

where *factor* is the value of the load factor to be applied. (Default value: 1.0)

5.9.3.10 LOAD_SET Argument

This argument changes the default load set number for element loads during either load definition or consistent external force formation.

Argument syntax:

LOAD_SET = *load_set*

where *load_set* is an integer load-set number. (Default value: 1)

5.9.3.11 MATRIX Argument

This argument sets the logical device index and dataset name for the factored system matrix.

Argument syntax:

MATRIX = *ldi, dataset_name*

where *ldi* is the logical device index for the file containing the matrix, and *dataset_name* is the name of the factored system matrix data set. (Default value: None)

5.9.3.12 MATRIX_UPDATE Argument

This argument sets the matrix-update mode for hierarchical adaptive refinement (relevant only for h_s-refinement).

Argument syntax:

MATRIX_UPDATE = {FULL | PARTIAL}

where FULL implies that the entire stiffness matrix is reformed for each new mesh, and where PARTIAL implies that only the updated-mesh contributions to the stiffness matrix are reformed for each new mesh. (Default value: FULL)

5.9.3.13 MAX_ITER Argument

This argument sets the maximum number of iterations allowed by an iterative linear equation solver (e.g., ITER). It is relevant only if SKY_PROCESSOR is set equal to the name of an iterative solver.

Argument syntax:

MAX_ITER = *max_iter*

where *max_iter* is the maximum number of iterations allowed. (Default value: 100)

5.9.3.14 MESH Argument

This argument sets the number of the mesh to analyze.

Argument syntax:

MESH = *mesh*

where *mesh* is the mesh number. (Default value: 0)

5.9.3.15 MTX_BUFFER_SIZE Argument

This argument sets the size of the memory buffer to be used for matrix factorization and solution by certain matrix solution processors.

Argument syntax:

MTX_BUFFER_SIZE = *mtx_buffer_size*

where *mtx_buffer_size* is the size of the buffer in terms of logical variables. (Default value: 500000)

5.9.3.16 REACTION Argument

This argument sets the compute reaction forces switch.

Argument syntax:

REACTION = *switch*

where *switch* is the option flag for computing the reaction forces. (Default value: <false>)

5.9.3.17 REFINE_TECHNIQUE Argument

This argument sets the refinement technique to be employed by the mesh refinement processor (REF*i*) specified via the REFINE_PROCESSOR argument.

Argument syntax:

REFINE_TECHNIQUE = *refine_technique*

where *refine_technique* is the name of the refinement technique. For example, in conjunction with processor REF1, the REFINE_TECHNIQUE argument might be set equal to "ht", "hc", "hs", or "p" (among others). See the documentation under specific REF*i* processors in Chapter 11 for details. (Default value: "ht")

5.9.3.18 RHS Argument

This argument sets the logical device index and dataset name for the right hand side (load) vector.

Argument syntax:

RHS = *ldi, dataset_name*

where *ldi* is the logical device index for the file containing the load vector, and *dataset_name* is the name of the load vector dataset. (Default value: None)

5.9.3.19 SOLN Argument

This argument sets the logical device index and dataset name for the solution vector.

Argument syntax:

SOLN = *ldi, dataset_name*

where *ldi* is the logical device index for the file containing the solution vector, and *dataset_name* is the name of the solution vector dataset. (Default value: None)

5.9.3.20 SKY_PROCESSOR Argument

Selects the matrix solution processor to be used for factoring and solving assembled linear equation systems.

Argument syntax:

SKY_PROCESSOR = *sky_processor*

where *sky_processor* is the name of the matrix solution processor. Current options are summarized below.

sky_processor	Description
SKY	Direct solution of skyline matrices by Crout decomposition (LDU) (Default)
SKYs	Direct and/or iterative solution of skyline matrices in conjunction with h_s-refinement only
ITER	Iterative solution of compact matrices by PCG algorithm
PVSOLV	Direct solution of skyline matrices optimized for vector computers.

Consult the appropriate processor section in Chapter 12 for more details.

5.9.3.21 SPEC_DISP Argument

This argument sets the logical device index and dataset name for the nodal specified displacement dataset.

Argument syntax:

SPEC_DISP = *ldi, dataset_name*

where *ldi* is the logical device index for the file containing the nodal table, and *dataset_name* is the name of the nodal specified displacements dataset. (Default value: None)

5.9.3.22 SOLVER_CONV_TOL Argument

This argument sets the convergence tolerance for the iterative linear equation solver, if one has been selected via the SKY_PROCESSOR argument.

Argument syntax:

SOLVER_CONV_TOL = *solver_conv_tol*

where *solver_conv_tol* is the convergence tolerance. (Default value: 1.e-5)

5.9.4 Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the matrix/vector algebra processor being used. These dataset requirements are documented in Chapter 12.

5.9.5 Current Limitations

SOLVE is a general purpose procedure and the only limitations on its use are dictated by the limitations of the ES*i* processor being employed. Refer to individual matrix/vector algebra processors in Chapter 12 for specific processor limitations.

5.9.6 Status and Error Messages

SOLVE does not print any status or error messages directly. All messages will be produced by the ES*i* processor being employed. Refer to individual matrix/vector algebra processors in Chapter 12 for specific processor messages.

5.9.7 Examples and Usage Guidelines

5.9.7.1 Example 1: Iterative Solution

```
*call SOLVE (    SKY_PROCESSOR      =  ITER                                ; --
                 SOLVER_CONV_TOL    =  1.0e-7                              ; --
                 MAX_ITER           =  1000                                ; --
                 ELT_MATRIX         =  2, E*.MATL_STIFFNESS...3            ; --
                 MATRIX             =  3, STRUCTURE.MATL_STIFFNESS...3     ; --
                 SOLN               =  1, NODAL.DISPLACEMENT.1.1.3         ; --
                 RHS                =  1, NODAL.EXT_FORCE.1..3             ; --
                 SPEC_DISP          =  1, NODAL.SPEC_DISP.1.0.3            ; --
                 MESH               =  3                                   ; --
                 LOAD_FACTOR        =  1.0                                  )
```

In this example, iterative solution for mesh 3 will be performed using the ITER processor. The assembled and factored matrix (in this case incomplete factorization of the COMPAXX format matrix) is in the standard system file (ldi=3 is associated with the *Case*.DBS file) and prescribed displacement contributions to the load vector will be added to the right hand side vector prior to solution using the element stiffness matrices from the standard element matrices file (ldi=2 is associated with the *Case*.DBE file).

The convergence tolerance for the iterative solution is set to 1.0e-7 and a maximum of 1000 iteration is allowed.

5.9.8 References

None.

5.10 Procedure STIFFNESS

5.10.1 General Description

This section describes the STIFFNESS Utility Procedure, which calls the ES utility procedure (FUNCTION = FORM STIFFNESS) to execute all element processors and types associated with a given model to compute element stiffness matrices, followed by an invocation of the appropriate assembly processor to assemble the system matrix.

5.10.2 Argument Summary

Procedure STIFFNESS may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 5.10-1.

Table 5.10-1 Procedure STIFFNESS Input Arguments

Argument	Default Value	Description
ASM_PROCESSOR	ASM	Matrix/vector assembly processor
ASM_STIFFNESS	—	Sets the default name of assembled stiffness dataset
AUTO_DRILL	<false>	Sets the default value of artificial drilling stiffness parameter
CONSTRAINT_SET	1	Constraint set number to be used for suppressing DOFs in the assembled system matrix prior to factorization
COROTATION	<false>	Sets the default element corotational option
DISPLACEMENT	—	Sets the default name of the nodal displacement dataset
ELT_STIFFNESS	—	The *ldi* and dataset name of the element stiffness matrices dataset
FIXED_FRAME	OFF	Fixed-frame option for hierarchical h_s-refinement
LDI_C	1	Sets the default *ldi* of computational database library
LDI_E	2	Sets the default *ldi* of element matrices database library
LDI_S	3	Sets the default *ldi* of system matrices database library
LOAD_FACTOR	1.0	Sets the default load factor to be applied to element loads
LOAD_SET	1	Sets the default load set number for element loads
MASS	DUMMY.MASS	
MATRIX_UPDATE	FULL	Matrix update option for hierarchical h_s-refinement
MESH	0	Sets the mesh number
MTX_BUFFER_SIZE	500000	Matrix buffer size for equation solving
NL_GEOM	<false>	Sets the default geometric nonlinearity option
NL_LOAD	<false>	Sets the default load nonlinearity option

Table 5.10-1 Procedure STIFFNESS Input Arguments (Continued)

Argument	Default Value	Description
REFINE_TECHNIQUE	ht	Mesh refinement technique (h_t => transition h)
ROTATION	—	Sets the default name of nodal rotation pseudovector dataset
SKY_PROCESSOR	SKY	Linear equation solver processor name
STEP	0	Sets/resets load- or time-step number
TYPE	TANG	Sets the default name of element stiffness dataset

5.10.3 Argument Definitions

In this subsection, the procedure arguments summarized Table 5.10-1 are defined in more detail. The arguments are listed alphabetically. Refer to Chapter 7, *Element Processors*, and Chapter 12, *Matrix/Vector Processors*, for details on the options.

5.10.3.1 ASM_PROCESSOR Argument

Selects the matrix assembly processor to be used for assembling element (stiffness/mass) matrices into corresponding system matrices.

Argument syntax:

ASM_PROCESSOR = *asm_processor*

where *asm_processor* is the name of the matrix assembly processor. Current options include ASM (for h_t and h_c types of mesh refinement) and ASMs (for h_s mesh refinement only). (Default value: ASM)

5.10.3.2 ASM_STIFFNESS Argument

This argument sets the *ldi* and dataset name of the assembled stiffness matrix.

Argument syntax:

ASM_STIFFNESS = *ldi, dataset_name*

where *ldi* is the logical device index associated with the system matrix file and *dataset_name* is the assembled system stiffness matrix dataset name. (Default value: None)

5.10.3.3 AUTO_DRILL Argument

Automatic drilling stiffness option. This option causes shell elements to add artificial drilling rotational stiffness to nodal DOFs that would otherwise be unstable computationally. See Section 2.10 and Chapter 7 for more information.

Argument syntax:

AUTO_DRILL = *option* [, *angle_tol*, *scale*_fac]

where

Parameter	Description
option	Automatic drilling stiffness switch: {<true> | <false>}. If <true>, certain shell element types will add artificial drilling stiffness to nodal DOFs that require stabilization. (Default value: <false>)
angle_tol	Angle tolerance to use for determining whether artificial drilling stiffness is needed at a given node. (Default value: depends on element type)
scale_fac	Scale factor determining magnitude of artificial drilling stiffness to be added by selected shell elements. (Default value: depends on element type)

AUTO_DRILL is not recommended for nonlinear analysis.

5.10.3.4 CONSTRAINT_SET Argument

This argument defines the constraint set number associated with the element, nodal, and system data. This number should appear as the second cycle number in names of all datasets.

Argument syntax:

CONSTRAINT_SET = *conset*

where:

Parameter	Description
conset	Constraint set number (Default value: 1)

5.10.3.5 COROTATION Argument

This argument sets the element corotational option for geometrically nonlinear analysis. The corotational capability is built in to the generic element processor (ES) and enables beam and shell

elements to be employed with arbitrarily large rotations (but small to moderate strains) even if the element strain-displacement relations do not intrinsically account for large rotations exactly.

Argument syntax:

COROTATION = *corotation_option*

where

corotation_option	Description
0 or <false>	Element corotation will not be used. (Default)
1	Basic element corotation will be used. This option is sufficient unless True-Newton iteration is being performed at the nonlinear solution procedure level.
2	Higher-order element corotation will be used. This option should be used only if True-Newton iteration has been selected at the nonlinear solution procedure level, and even then may provide only marginal improvement in nonlinear convergence over option 1. It adds additional terms to the tangent stiffness matrix that render it more consistent.

5.10.3.6 DISPLACEMENT Argument

This argument sets the name of the nodal displacement dataset.

Argument syntax:

DISPLACEMENT = *ds_name*

where *ds_name* is the nodal displacement dataset name.
(Default value: NODAL.DISPLACEMENT.1.1)

5.10.3.7 ELT_STIFFNESS Argument

This argument sets the *ldi* and dataset name of the element stiffness matrices dataset.

Argument syntax:

ELT_STIFFNESS = *ldi, dataset_name*

where *ldi* is the logical device index associated with the element matrices file and *dataset_name* is the element stiffness matrix dataset name. (Default value: None)

5.10.3.8 FIXED_FRAME Argument

Sets a flag that is relevant only for h_s-refinement.

Argument syntax:

FIXED_FRAME = {<true> | <false>}

Do not change the default setting without the advice of a COMET-AR expert. (Default value: <false>)

5.10.3.9 LDI_C Argument

This argument sets the logical device index associated with the main COMET-AR database file, which must exist before calling L_STATIC_1 and must be named *Case*.DBC.

Argument syntax:

LDI_C = *ldi_c*

where *ldi_c* is the logical device index (a positive integer) of the *Case*.DBC file. (Default value: 1)

5.10.3.10 LDI_E Argument

This argument sets the logical device index associated with the element matrix database file, typically named *Case*.DBE.

Argument syntax:

LDI_E = *ldi_e*

where *ldi_e* is the logical device index (a positive integer) of the *Case*.DBE file. If *ldi_e* is not equal to *ldi_c* (see the LDI_C argument) then all element matrices (e.g., stiffness and mass) for the current mesh will be stored on a separate *Case*.DBE file. If *ldi_e* = *ldi_c*, then all element matrices will be stored on the *Case*.DBC file, i.e., a separate *Case*.DBE file will not be created. (Default value: 2)

If a separate *Case*.DBE file is created, it will be deleted and re-created with each new adaptive mesh.

5.10.3.11 LDI_S Argument

This argument sets the logical device index associated with the system matrix database file, typically named *Case*.DBS.

Argument syntax:

LDI_S = *ldi_s*

where *ldi_s* is the logical device index (a positive integer) of the *Case*.DBS file. If *ldi_s* is not equal to *ldi_c* (see the LDI_C argument) then all system matrices (e.g., stiffness and mass) for the current mesh will be stored on a separate *Case*.DBS file. If *ldi_s* = *ldi_c*, then all system matrices will be stored on the *Case*.DBC file, i.e., a separate *Case*.DBS file will not be created. (Default value: 3)

If a separate *Case*.DBS file is created, it will be deleted and re-created with each new adaptive mesh.

5.10.3.12 LOAD_FACTOR Argument

This argument changes the default load factor to be applied to all element loads.

Argument syntax:

LOAD_FACTOR = *load_factor*

where *load_factor* is a floating-point scale factor. (Default value: 1.0)

5.10.3.13 LOAD_SET Argument

This argument changes the default load set number for element loads during either load definition or consistent external force formation.

Argument syntax:

LOAD_SET = *load_set*

where *load_set* is an integer load-set number. (Default value: 1)

5.10.3.14 MATRIX_UPDATE Argument

This argument sets the matrix-update mode for hierarchical adaptive refinement (relevant only for h_s-refinement).

Argument syntax;

MATRIX_UPDATE = {FULL | PARTIAL}

where FULL implies that the entire stiffness matrix is reformed for each new mesh, and where PARTIAL implies that only the updated-mesh contributions to the stiffness matrix are reformed for each new mesh. (Default value: FULL)

5.10.3.15 MESH Argument

This argument changes the default mesh number used in all dataset names (unless otherwise specified via a separate dataset command).

Argument syntax:

MESH = *mesh*

where *mesh* is an integer number, typically set to the current mesh number. (Default value: 0)

5.10.3.16 MTX_BUFFER_SIZE Argument

This argument sets the size of the memory buffer to be used for matrix factorization and solution by certain matrix solution processors.

Argument syntax:

MTX_BUFFER_SIZE = *mtx_buffer_size*

where *mtx_buffer_size* is the size of the buffer in terms of logical variables. (Default value: 500000)

5.10.3.17 NL_GEOM Argument

This argument changes the default geometric nonlinearity option. It is often used in conjunction with the COROTATION command.

Argument syntax:

NL_GEOM = *nl_geom_option*

where

nl_geom_option	Description
0 or <false>	The analysis is geometrically linear; linear element strain-displacement relations will be employed and element corotational will be disregarded. (Default)
1	The analysis is geometrically nonlinear, but only linear element strain-displacement relations will be used. With this option geometric nonlinearity must be accounted for via element corotation (see the COROTATION command), which for many beam/shell element types is not as accurate as option 2.
2	The analysis is geometrically nonlinear, and nonlinear element strain-displacement relations will be used. Element corotation may or not be selected with this option. For many beam/shell element types, nonlinear element strain-displacement relations enhance corotation, making it more accurate for a given mesh and rotation magnitude.

5.10.3.18 NL_LOAD Argument

This argument changes the default load nonlinearity option. It affects whether "live" loads are to be processed as part of the external force vector or the tangent stiffness matrix.

Argument syntax:

NL_LOAD = *nl_load_option*

where

nl__load_option	Description
0 or <false>	Ignore load nonlinearity (i.e., displacement dependence). Only displacement-independent ("dead") external loads are to be processed in the following FORM FORCE or FORM STIFFNESS command. (Default)
1	Include load nonlinearity. Only displacement-dependent ("live") external loads are to be processed in the following FORM FORCE or FORM STIFFNESS command.

5.10.3.19 REFINE_TECHNIQUE Argument

This argument sets the refinement technique to be employed by the mesh refinement processor (REF*i*) specified via the REFINE_PROCESSOR argument.

Argument syntax:

REFINE_TECHNIQUE = *refine_technique*

where *refine_technique* is the name of the refinement technique. For example, in conjunction with processor REF1, the REFINE_TECHNIQUE argument might be set equal to h_t, h_c, h_s, or p (among others). See documentation under specific REF*i* processors for details. (Default value: h_t)

5.10.3.20 ROTATION Argument

This argument changes the default name of the nodal rotation (pseudovector) dataset.

Argument syntax:

ROTATION = *ds_name*

where *ds_name* is the new dataset name. (Default value: NODAL.ROTATION.1.1)

5.10.3.21 SKY_PROCESSOR Argument

Selects the matrix solution processor to be used for factoring and solving assembled linear equation systems.

Argument syntax:

SKY_PROCESSOR = *sky_processor*

where *sky_processor* is the name of the matrix solution processor. Current options are summarized below.

sky_processor	Description
SKY	Direct solution of skyline matrices by Crout decomposition (LDU) (Default)
SKYs	Direct and/or iterative solution of skyline matrices in conjunction with h_s-refinement only
ITER	Iterative solution of compact matrices by PCG algorithm
PVSOLV	Direct solution of skyline matrices optimized for vector computers.

Consult Chapter 12 for more details.

5.10.3.22 STEP Argument

This argument changes the default load- or time-step number used in many solution dataset names (unless otherwise specified via a separate dataset command).

Argument syntax:

STEP = *step*

where *step* is an integer number, typically set to the current step number. (Default value: 0)

5.10.3.23 TYPE Argument

This argument sets the type of stiffness matrix to be computed.

Argument syntax:

TYPE = *type*

where *type* is the type of stiffness to be computed (TANG, GEOM, or MATL for tangent, geometry or material stiffnesses, respectively). (Default value: TANG)

5.10.4 Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the ES*i* processor being used and the FUNCTION argument. These dataset requirements are documented in Chapters 7 and 12.

5.10.5 Current Limitations

STIFFNESS is a general purpose procedure and the only limitations on its usage are dictated by the limitations of the ES*i* and matrix/vector algebra processors being employed. Refer to individual processors in Chapters 7 and 12 for specific processor limitations.

5.10.6 Status and Error Messages

STIFFNESS does not print any status or error messages directly. All messages will be produced by the ES*i* and matrix/vector algebra processors being employed. Refer to individual processors in Chapters 7 and 12 for specific processor messages.

5.10.7 Examples and Usage Guidelines

5.10.7.1 Example 1: Material Stiffness Formation and Assembly in COMPAXX Format

```
*call STIFFNESS (  TYPE            =  MATL                                  ; --
                   ELT_STIFFNESS   =  2, E*.MATL_STIFFNESS...1              ; --
                   NL_MATL         =  <false>                               ; --
                   NL_GEOM         =  <false>                               ; --
                   SKY_PROCESSOR   =  ITER                                  ; --
                   ASM_STIFFNESS   =  3, STRUCTURE.MATL_STIFFNESS...2       ; --
                   MESH            =  2                                     ; --
                   ASM_PROCESSOR   =  ASM                                    )
```

In this example, the formation of element linear material stiffnesses is requested for mesh 2. The element stiffness matrices will be stored in 2, *EltNam*.STIFFNESS...2. The assembled matrix in COMPAXX format, as required by the ITER processor, will be stored in a dataset named 3, STRUCTURE.MATL_STIFFNESS...2.

5.10.8 References

None.

5.11 Procedure STRESS

5.11.1 General Description

This section describes the STRESS Utility Procedure which calls the ES utility procedure (FUNCTION = FORM STRESS) to executes all element processors associated with a given model to recover element stresses from a given displacement solution.

5.11.2 Argument Summary

Procedure STRESS may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 5.11-1.

Table 5.11-1 Procedure STRESS Input Arguments

Argument	Default Value	Description
COROTATION	<false>	Sets the default element corotational option
DIRECTION	0	Sets the default stress/strain output coordinate system
DISPLACEMENT	1, NODAL.DISPLACEMENT.1.1	Sets the default name of the nodal displacement dataset
LOCATION	INTEG_PTS	Sets the default stress/strain output locations
MESH	0	Sets the mesh number
NL_GEOM	<false>	Sets the default geometric nonlinearity option
ROTATION		Sets the default name of the nodal rotation pseudovector dataset
SE_TOT	<false>	
STEP	0	Sets/resets load- or time-step number
STRAIN	1, E*.STRAIN.1.1	Sets the default ldi and name of the element strain dataset
STRAIN_ENERGY	1, E*.STRAIN_ENERGY.1.1	Sets the default ldi and name of the element strain energy dataset
STRESS	1, E*.STRESS.1.1	Sets the default ldi and name of the element stress dataset

5.11.3 Argument Definitions

In this subsection, the procedure arguments summarized in Table 5.11-1 are defined in more detail. The arguments are listed alphabetically. Refer to Chapter 7 for details on the options.

5.11.3.1 COROTATION Argument

This argument sets the default element corotational option for geometrically nonlinear analysis. The corotational capability is built in to the generic element processor (ES) and enables beam and shell elements to be employed with arbitrarily large rotations (but small to moderate strains), even if the element strain-displacement relations do not intrinsically account for large rotations exactly.

Argument syntax:

COROTATION = *corotation_option*

where

corotation_option	Description
0 or <false>	Element corotation will not be used. (Default)
1	Basic element corotation will be used. This option is sufficient unless True-Newton iteration is being performed at the nonlinear solution procedure level.
2	Higher-order element corotation will be used. This option should be used only if True-Newton iteration has been selected at the nonlinear solution procedure level; and even then may provide only marginal improvement in nonlinear convergence over option 1. It adds additional terms to the tangent stiffness matrix that render it more consistent.

5.11.3.2 DIRECTION Argument

This argument changes the default stress or strain direction option prior to use of the FORM STRAIN, FORM STRESS, FORM FORCE/INT, or FORM FORCE/RES FUNCTION arguments. (Default value: 0)

Argument syntax:

DIRECTION = *str_direction*

where

str_direction	Description
ELEMENT or 0	Use element local (integration point) coordinate system, x_l, y_l, z_l, as stress/strain output system: x_s, y_s, z_s. (Default)
GLOBAL { X | Y | Z }	The stress/strain output x_s axis is parallel to the global x_g, y_g, or z_g axis if X, Y, or Z, respectively, is used in the subcommand. The stress/strain output z_s axis is parallel to the local element normal axis for shell elements, otherwise it is obtained by permutating the global axes. The stress/strain output y_s axis is defined by the right-hand-rule.

str_direction	Description
FAB_DIR	Use the local material-fabrication coordinate system, x_f, y_f, z_f, as the stress/strain output system, x_s, y_s, z_s.

5.11.3.3 DISPLACEMENT Argument

This argument changes the default name of the nodal displacement dataset.

Argument syntax:

DISPLACEMENT = *ds_name*

where *ds_name* is the nodal displacement dataset name.
(Default value: NODAL.DISPLACEMENT.1.1)

5.11.3.4 LOCATION Argument

This argument changes the default stress, strain, or strain-energy location option prior to use of the FORM STRAIN, FORM STRESS, FORM STRAIN_ENERGY, FORM FORCE/INT, or FORM FORCE/RES FUNCTION arguments. (Default value: INTEG_PTS)

Argument syntax:

LOCATION = *str_location*

where

str_location	Description
INTEG_PTS	Element stresses, strains, or strain-energy densities will be evaluated at element integration points and stored in the STR attribute of the specified EST dataset.
NODES	Element stresses, strains, or strain-energy densities will be evaluated at integration points, then extrapolated and stored at element nodes in the STRNOD attribute of the specified EST dataset.
CENTROIDS	Element stresses, strains, or strain-energy densities will first be evaluated at the element integration points, then averaged and stored at element centroids in the STRCEN attribute of the specified EST dataset. (If one of the element's integration points coincides with the centroid, the value computed there will be output rather than an average integration-point value.)

5.11.3.5 MESH Argument

This argument changes the default mesh number used in all dataset names (unless otherwise specified via a separate dataset command).

Argument syntax:

MESH = *mesh*

where *mesh* is an integer number, typically set to the current mesh number. (Default value: 0)

5.11.3.6 NL_GEOM Argument

This argument changes the default geometric nonlinearity option. It is often used in conjunction with the COROTATION command.

Argument syntax:

NL_GEOM = *nl_geom_option*

where

nl_geom_option	Description
0 or <false>	The analysis is geometrically linear; linear element strain-displacement relations will be employed, and element corotational will be disregarded. (Default)
1	The analysis is geometrically nonlinear, but only linear element strain-displacement relations will be used. With this option geometric nonlinearity must be accounted for via element corotation (see the COROTATION command), which for many beam/shell element types is not as accurate as option 2.
2	The analysis is geometrically nonlinear, and nonlinear element strain-displacement relations will be used. Element corotation may or not be selected with this option. For many beam/shell element types, nonlinear element strain-displacement relations enhances corotation, making it more accurate for a given mesh and rotation magnitude.

5.11.3.7 ROTATION Argument

This argument changes the default name of the nodal rotation (pseudovector) dataset.

Argument syntax:

ROTATION = *ds_name*

where *ds_name* is the new dataset name. (Default value: NODAL.ROTATION.1.1)

5.11.3.8 STEP Argument

This argument defines the solution step number associated with the element solution data for which error estimates are to be computed. This number appears as the first cycle number in names of all element solution datasets, e.g., STRESS, STRAIN, and STRAIN_ENERGY (relevant only for nonlinear static analysis).

Argument syntax:

STEP = *step*

where

Parameter	Description
step	Solution step number. (Default value: None)

5.11.3.9 STRAIN Argument

This argument changes the default name of the element strain dataset before using the FORM STRAIN command. It also causes strains to be output to the database by the FORM STRESS, FORM FORCE/INT, or FORM FORCE/RES FUNCTION arguments.

Argument syntax:

STRAIN = *ldi, ds_name*

where *ldi* is the new logical device index and *ds_name* is the new dataset name. (Default value: 1, *EltName*.STRAIN.1.1.*mesh*)

5.11.3.10 STRAIN_ENERGY Argument

This argument changes the default name of the element strain-energy density dataset before using the FORM STRAIN_ENERGY command. It also causes strain-energy densities to be output to the database by the FORM STRESS, FORM FORCE/RES, or FORM FORCE/INT FUNCTION arguments.

Argument syntax:

STRAIN_ENERGY = *ldi, ds_name*

where *ldi* is the new logical device index and *ds_name* is the new dataset name. (Default value: 1, *EltName*.STRAIN_ENERGY.1.1.*mesh*)

5.11.3.11 STRESS Argument

This argument changes the default *ldi* and name of the element stress dataset before using the FORM STRESS command. It also causes strains to be output to the database by the FORM FORCE/INT or FORM FORCE/RES FUNCTION arguments.

Argument syntax:

```
STRESS = ldi, ds_name
```

where *ldi* is the new logical device index and *ds_name* is the new dataset name. (Default value: 1, *EltName*.STRESS.1.1.*mesh*)

5.11.4 Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the ES*i* processor used and the FUNCTION argument. These dataset requirements are documented in Chapter 7.

5.11.5 Current Limitations

STRESS is a general purpose procedure and the only limitations on its usage are dictated by the limitations of the ES*i* processor being employed. Refer to individual ES*i* processors in Chapter 7 for specific processor limitations.

5.11.6 Status and Error Messages

STRESS does not print any status or error messages directly. All messages will be produced by the ES*i* processor being employed. Refer to individual ES*i* processors in Chapter 7 for specific processor messages.

5.11.7 Examples and Usage Guidelines

5.11.7.1 Example 1: Recover Element Stresses at Integration Points

```
*call STRESS (     MESH      =  3          )
```

In this example, a complete stress recovery for mesh 3 will be performed. Element stresses, strains, and strain energies will be stored in the 1, *EltNam*.STRESS/STRAIN/STRAIN_ENERGY.1.1.2 datasets.

5.11.8 References

None.

5.12 Procedure MASS

5.12.1 General Description

The MASS procedure is a utility procedure typically called by dynamic analysis procedures (such as L_DYNAMIC_1) to compute and/or assemble a system mass matrix, lumped or consistent, from element and/or nodal (lumped) mass contributions.

5.12.2 Argument Summary

Procedure MASS may be invoked with the COMET-AR *CALL directive, employing the arguments summarized in Table 5.12-1.

Table 5.12-1 Procedure MASS Input Arguments

Argument	Default Value	Description
ASM_MASS	STRUCTURE.MASS	Name of assembled system mass matrix (for consistent mass matrices only)
ASM_PROCESSOR	ASM	Name of assembly processor to use
CONSTRAINT_SET	1	Constraint set number to be used for suppressing DOFs in the assembled system matrix prior to factorization
ELT_MASS	E*.MASS	Name of assembled diagonal/lumped mass matrix (stored as a nodal vector table (NVT).
LDI_C	1	Sets the default *ldi* of computational database library
LDI_E	2	Sets the default *ldi* of the element matrices database library
LDI_S	3	Sets the default *ldi* of the system matrix database library
TYPE	CONSISTENT	Type of assembled mass matrix: LUMPED or CONSISTENT

5.12.3 Argument Definitions

In this subsection, the procedure arguments summarized Table 5.12-1 are defined in more detail. The arguments are listed alphabetically. Refer to Chapter 7 and Chapter 12 for details on the specific element and assembly processor options.

5.12.3.1 ASM_MASS Argument

Name of the assembled mass matrix dataset.

Argument syntax:

ASM_MASS = *asm_mass*

where *asm_mass* is the name of the assembled mass matrix dataset. (Default: STRUCTURE.MASS) Currently, ASM_MASS is used as the name of the output dataset only if TYPE=CONSISTENT; otherwise, ELT_MASS is used as the name of the lumped (i.e., diagonal) mass matrix.

5.12.3.2 ASM_PROCESSOR Argument

Selects the matrix assembly processor to be used for assembling element mass matrices into a corresponding system matrix.

Argument syntax:

ASM_PROCESSOR = *asm_processor*

where *asm_processor* is the name of the matrix assembly processor. Current options include ASM (for h_t and h_c types of mesh refinement) and ASMs (for h_s mesh refinement only). (Default value: ASM)

5.12.3.3 CONSTRAINT_SET Argument

This argument defines the constraint set number associated with the element, nodal, and system data. This number should appear as the second cycle number in names of all datasets.

Argument syntax:

CONSTRAINT_SET = *conset*

where:

Parameter	Description
conset	Constraint set number (Default value: 1)

5.12.3.4 ELT_MASS Argument

This argument represents the name to be used for the output assembled mass matrix, if the matrix type (see TYPE argument below) is DIAGONAL.

Argument syntax:

ELT_MASS = *Elt_Mass*

where *Elt_Mass* is the name of the assembled diagonal mass matrix dataset to be output. (Default: NODAL.DIAG_MASS)

5.12.3.5 LDI_C Argument

This argument sets the logical device index associated with the main COMET-AR database file, which must exist before calling L_STATIC_1 and must be named *Case*.DBC.

Argument syntax:

LDI_C = *ldi_c*

where *ldi_c* is the logical device index (a positive integer) of the *Case*.DBC file. (Default value: 1)

5.12.3.6 LDI_E Argument

This argument sets the logical device index associated with the element matrix database file, typically named *Case*.DBE. This argument is relevant only for consistent mass matrices.

Argument syntax:

LDI_E = *ldi_e*

where *ldi_e* is the logical device index (a positive integer) of the *Case*.DBE file. If *ldi_e* is not equal to *ldi_c* (see the LDI_C argument), then all element mass matrices for the current mesh will be stored on a separate *Case*.DBE file. If *ldi_e* = *ldi_c*, then all element mass matrices will be stored on the *Case*.DBC file, i.e., a separate *Case*.DBE file will not be created. (Default value: 2)

If a separate *Case*.DBE file is created, it will be deleted and re-created with each new adaptive mesh.

5.12.3.7 LDI_S Argument

This argument sets the logical device index associated with the system matrix database file, typically named *Case*.DBS. The argument is relevant only for consistent mass matrices (for diagonal mass matrices, the assembled matrix, which is really a nodal vector, is stored in the library associated with LDI_C).

Argument syntax:

LDI_S = *ldi_s*

where *ldi_s* is the logical device index (a positive integer) of the *Case*.DBS file. If *ldi_s* is not equal to *ldi_c* (see the LDI_C argument) then all system mass matrices for the current mesh will be stored on a separate *Case*.DBS file. If *ldi_s* = *ldi_c*, then all system matrices will be stored on the *Case*.DBC file, i.e., a separate *Case*.DBS file will not be created. (Default value:3)

If a separate *Case*.DBS file is created, it will be deleted and re-created with each new adaptive mesh.

5.12.3.8 TYPE Argument

This argument sets the type of mass matrix to be computed.

Argument syntax:

TYPE = *type*

where *type* is the type of stiffness to be computed. Current options are: CONSISTENT and DIAGONAL. (Default: CONSISTENT)

5.12.4 Database Input/Output Summary

A model definition database is required as input for the MASS procedure (see Chapter 2, *Model Definition Procedures*). After invoking the MASS procedure, either a consistent mass matrix will be deposited in the data library associated with LDI_S (and element mass matrices will be deposited in the data library associated with LDI_E), or a diagonal mass matrix (in nodal vector format) will be deposited in the library associated with LDI_C. In additional to the usual model input data, a "NODAL.MASS" dataset may also be defined by the user, via processor NODAL. This dataset contains user-specified lumped nodal contributions to the mass matrix; the MASS procedure adds this lumped nodal mass dataset to the element mass matrices when creating the final assembled mass matrix, whether consistent or diagonal.

5.12.4.1 Input Datasets

Table 5.12-2 contains a list of datasets required (unless otherwise stated) as input by procedure MASS. All of these datasets must be resident in the main COMET-AR database (*Case*.DBC, where *Case* is the specific problem name).

Table 5.12-2 Input Datasets Required by Procedure MASS

Dataset	File	Description
CSM.SUMMARY...*mesh*	LDI_C	Model summary for the analyzed mesh
EltName.DEFINITION...*mesh*	LDI_C	Element definition for the analyzed mesh
EltName.FABRICATION...*mesh*	LDI_C	Element fabrication pointers for the analyzed mesh
EltName.GEOMETRY...*mesh*	LDI_C	Element solid-model geometry for the analyzed mesh
EltName.INTERPOLATION...*mesh*	LDI_C	Element interpolation data for the analyzed mesh
NODAL.COORDINATE...*mesh*	LDI_C	Nodal coordinates for the analyzed mesh
NODAL.DOF..*conset.mesh*	LDI_C	Nodal DOF Table for the analyzed mesh.
NODAL.TRANSFORMATION...*mesh*	LDI_C	Nodal transformations between global and computational frames for the analyzed mesh
NODAL.MASS.*ldcase..mesh*	LDI_C	Nodal lumped masses to be added to the element mass matrices during assembly

5.12.4.2 Output Datasets

Table 5.12-3 contains a list of datasets that may be created in the database by procedure MASS.

Table 5.12-3 Output Datasets Produced by Procedure MASS

Dataset	Class	File	Description
[ASM_MASS]	SMT	LDI_S	Assembled system mass matrix (Output only if TYPE=CONSISTENT)
E*.MASS	EMT	LDI_E	Element mass matrices (Output only if TYPE=CONSISTENT)
[ELT_MASS]	NVT	LDI_C	Assembled diagonal mass matrix (Output only if TYPE=DIAGONAL)

For details on the contents of any of the above datasets, refer to Chapter 15, *Database Summary.*

5.12.5 Current Limitations

Procedure MASS will not generate a DIAGONAL mass matrix if there are any multi-point constraints (MPCs). This is because MPCs typically induce coupling terms that would not be properly accounted for. When MPCs are present, the user should employ a consistent mass matrix.

5.12.6 Status and Error Messages

None.

5.12.7 Examples and Usage Guidelines

5.12.7.1 Example 1: Diagonal Mass Matrix Formation

```
*call MASS (        TYPE            =  DIAGONALL                    ; --
                    ELT_MASS        =  NODAL.DIAG_MASS                --
                                                                      )
```

In this example, a diagonal mass matrix (NVT dataset) is stored in a dataset called NODAL.DIAG_MASS. The assembly processor (ASM_PROCESSOR) is irrelevant for such cases, as the diagonal mass matrix is assembled by vector addition, via processor VEC. If a user-specified lumped nodal mass dataset (which must be called NODAL.MASS) is present, the dataset will automatically be added into the assembled diagonal mass matrix by procedure MASS.

5.12.8 References

None.

Part III

PROCESSORS

In this part of the COMET-AR User's Manual, we describe available Fortran level processors (i.e., independently executable command/database-driven modules) that may be invoked by the user for a variety of functions, including pre-processing, analysis, and post-processing. While any of these processors may be employed interactively, they are typically invoked indirectly and automatically via COMET-AR procedures (see Part I). An exception to this is processor ARGx, a graphical post-processor that is strictly interactive.

Chapter 6 Pre-Processors

6.1 Overview

In this chapter, various pre-processors implemented in COMET-AR are described. These processors are used primarily for model definition as indicated in Chapter 2, *Model Definition Procedures*. A summary of currently available pre-processors within this chapter is given in Table 6.1-1.

Table 6.1-1 Outline of Chapter Chapter 6: Pre-Processors

Section	Processor	Function
6.2	AUS	Nodal force/displacement tabulation
6.3	COP	Nodal constraint definition
6.4	GCP	Generic constitutive processor
6.5	GEP	Generic element processor
6.6	PST	PATRAN-to-COMET-AR conversion
6.7	REDO	Reformatting of TAB and AUS datasets
6.8	RENO	Node/bandwidth renumbering; geometric algorithm
6.9	RSEQ	Node/bandwidth renumbering; various algorithms
6.10	TAB	Nodal coordinate/transformation tabulation

6.2 Processor AUS (Nodal Force Tabulation)

6.2.1 General Description

Processor AUS is used by COMET-AR to define nodal loads, i.e., point forces and/or nodal specified displacements. The SYSVEC subprocessor constructs system vector data tables which are subsequently translated into High Level Database (HDB) objects by the processor REDO, as described in Section 6.7. Detailed information about the SYSVEC subprocessor and command structure is contained in the remainder of this section.

6.2.2 Command Summary

Processor AUS follows the SPAR command syntax as described in Reference [1]. A summary of valid commands is given in Table 6.2-1.

Table 6.2-1 Processor AUS Command Summary

Command Name	Function
SYSVEC	Create or modify SYSVEC format datasets

6.2.3 Command Definitions

6.2.3.1 SYSVEC Command

The SYSVEC subprocessor is used to create and modify datasets in SYSVEC format. The command format for the SYSVEC subprocessor is:

SYSVEC[,U]: N1, N2, n3, n4

I = $i^1, i^2, \ldots, i^6$

J = $j_{beg}, j_{end}, j_{inc}$

$e^{i1}_{jbeg}, e^{i2}_{jbeg}, \ldots, e^{i6}_{jbeg}$

$e^{i1}_{(jbeg+jinc)}, e^{i2}_{(jbeg+jinc)}, \ldots, e^{i6}_{(jbeg+jinc)}$

...

where

Parameter	Description
U	Transfers the SYSVEC subprocessor into update mode, allowing for modification of an existing SYSVEC dataset.
N1,N2,n3,n4	Names to be used in the construction of the SYSVEC dataset. N1 and N2 are character input and n3, and n4 are integers. The resulting dataset will be named N1.N2.n3.n4.
I = $i^1, i^2, \ldots, i^6$	Row numbers for application of forces or specified displacements. $i^k = 1, 2,$ or 3 always indicates a direction-i^k displacement or force component; $i^k = 4, 5, 6$ indicates a rotation in radians or moment about axis $i^k - 3$
J = $j_{beg}, j_{end}, j_{inc}$	Column numbers for application of forces or specified displacements in loop limit format.
$e^{i^1}_{jbeg}, e^{i^2}_{jbeg}, \ldots, e^{i^6}_{jbeg}$	Load/Displacement values

The command runstream:

```
RUN AUS
   SYSVEC: APPL FORC 1
                  I=3
                  J=9,10: -1.0, -1.0
STOP
```

creates a dataset named APPL.FORC.1.1 with (number of active degrees of freedom) rows and (total number of nodes) columns. All entries will be zero except for the z-direction forces for nodes 9 and 10 which will each have a value of -1.0.

6.2.3.2 Input Datasets

A summary of input datasets used by Processor AUS is given in Table 6.2-2.

Table 6.2-2 Processor AUS Input Datasets

Dataset/Attribute	Contents
JDF1.BTAB.1.8	Dataset containing the total number of nodes in the model. Created by the TAB Processor.

6.2.3.3 Output Datasets

A summary of output datasets created by Processor AUS is given in Table 6.2-3.

Table 6.2-3 Processor AUS Output Datasets

Dataset/Attribute	Contents
APPL.FORC.1.1	Nodal point forces
APPL.MOTI.1.1	Nodal specified displacements

6.2.4 Limitations

AUS is an internal processor within the COMET-AR macroprocessor. As such, there is a blank common limit which is installation dependent. SYSVEC will notify the user if the memory required for processing the commands is insufficient, in which case you will need to increase the blank common of the executable.

6.2.5 Error Messages

The SYSVEC subprocessor checks to ensure that there is sufficient memory available to perform the requested function. In addition to these errors, input errors are reported by SYSVEC. These errors are summarized below.

Command	Error Message and User Response
SYSVEC	INPUT DATA ERROR — Fatal error; User input is in error.

6.2.6 Examples and Usage Guidelines

It is important that the computational GAL library (*ldi*) contain the dataset JDF1.BTAB.1.8 produced as a result of the START command in TAB. Any SYSVEC dataset operated on in processor AUS must correspond to the JDF1.BTAB.1.8 dataset present in this *ldi.*

The command runstream presented below creates the applied force dataset APPL.FORC.1.1 with a force applied in the global z direction to node 4, with a value of -1.0.

```
RUN AUS
        SYSVEC: APPL FORC 1
                i=3: j=4: -1.0
STOP
```

The command runstream presented below creates the specified displacement dataset APPL.MOTI.1.1 with a displacement of -1.0, applied in the global x direction to nodes <np1> through <nnt> as defined via the CLAMP do loop and macrosymbols.

```
RUN AUS
        SYSVEC: APPL MOTI 1
        *do $i = <np1>,<nnt>,1
                i=1: j=<$i>: -1.0
        *enddo
STOP
```

6.2.7 References

[1] Stewart, C. B., ed., *The Computational Structural Mechanics Testbed User's Manual*, NASA TM-100644, 1989.

6.3 Processor COP (Constraint Processor)

6.3.1 General Description

Processor COP is used to define and store the degrees-of-freedom (DOFs) and their constraints for each node point of a COMET-AR model. This information constitutes what is called a Nodal DOF Table (NDT data object), the logical view of which is described in Reference [1].

The COP processor is used to form an NDT data object for any analysis using the ASM, SKY, and/or related COMET-AR processors, all of which operate with DOF-oriented (as opposed to nodally-oriented) system matrices and vectors.

An NDT data object includes a table that indicates the number of freedoms that are associated with each node point and the type of freedom that is associated with each direction at each node point of the model. In the current version of COP, any given freedom may have one of the following constraint status indications:

FREE	unconstrained, independent DOF
ZERO	SPCz (Single-Point-Constrained) DOF, the value of which is zero
NONZERO	SPCnz DOF, the value of which is a specified constant
MPC	MPC (Multi-Point-Constrained) dependent DOF, to be expressed in terms of zero or more independent DOFs via a linear multi-point constraint relation and eliminated from the equation system

An NDT data object also includes information required to describe any SPCs and/or MPCs to which the model may be subjected. The present version of COP assumes that each dependent degree of freedom u_d to be eliminated from the equation system is expressed in terms of N_{id} independent freedoms u_i through a linear multi-point constraint relation of the form:

$$u_d = \sum_{i=1}^{N_{id}} \left[C_{di} \times u_i \right] + \alpha_d$$

where the C_{di} are proportionality constants that relate u_d to the N_{id} independent freedoms, and where α_d is the so-called intercept constant for the relation. COP enables the user to identify specific DOF u_d that are linearly dependent on (independent) DOF u_i (and/or α_d), and to specify the weighting coefficients C_{di} for the freedoms on which they are dependent. COP makes no assumptions about how single- or multi-point constraints are enforced; it passes this information along to other processors that know what to do with it.

An NDT data object also includes a table giving the equation number assigned to each freedom of the model. Normally each independent DOF has an equation number assigned to it, but COP permits you to override this convention.

The COP processor also performs two essential vector-transformation operations. Given an input vector that contains information only for the computational (independent) degrees of freedom, a specific NDT data object, and (optionally) other information, COP can expand the input vector into a nodally-oriented Nodal Vector Table (NVT) data object, calculating the values of any dependent freedoms with the multi-point constraint information in the NDT data object, and imposing any (ZERO and/or NONZERO) SPCs that may be imposed. Given an input vector that is stored in a nodally-oriented NVT data object, COP can also contract the information, extracting the independent DOF values contained therein to form a vector that is stored in the System Vector Table (SVT) data object form used by ASM, SKY, and other COMET-AR processors.

6.3.2 Processor Command Summary

The user must employ CLIP directives to communicate directly with GAL database files and do the general bookkeeping, branching, and arithmetic operations that are described in Reference [2].

The COP-specific commands that enable the user to operate on a database-resident NDT data object, or to use this information to contract or expand system vectors, are described here. Some of these commands facilitate the construction of a new Nodal DOF Table, or retrieve an existing NDT data object from its GAL database location. Others modify an NDT data object, changing the constraint status indications (states) of freedoms (by applying single- and/or multi-point constraints, suppressing or allowing the assignment of equation numbers for various freedom states, imposing an externally-determined nodal ordering when equation numbers are assigned, etc.). Other commands save the NDT data object on a GAL library file, and/or display it. Still other COP-specific commands facilitate the transformation of system vectors from the compressed, DOF-oriented SVT data object form that is used by ASM, SKY, and other COMET-AR processors to the nodally-oriented NVT-data-object form[1] used by other COMET-AR processors or vice versa.

The remainder of this section concentrates on these COP-specific commands. The current version of COP accepts the commands listed in Table 6.3-1.

Table 6.3-1 Processor COP Command Summary

Command Name	Function
MODEL	Specify a Complete Model Summary (CSM data object) dataset
SELECT	Retrieve an initial NDT data object from a GAL database, or construct a new one
SEQUENCE	Specify nodal-ordering information
RESET	Reset a program-control parameter
DOF_SUPPRESS	Set constraint-status indicators in a designated NDT data object to reflect DOF suppressions indicated in a given DOF-suppression table

1. The System Vector Table (SVT data object) data structure used here replaces the DOFVEC format used by earlier versions of ASM, COP, and SKY; and the Nodal Vector Table (NVT data object) structure replaces the SYSVEC format used by the Testbed and its older relatives. Both of these object-oriented structures are described in Reference [1].

Table 6.3-1 Processor COP Command Summary (Continued)

Command Name	Function
CONSTRAIN	Transfer control to the CONSTRAIN sub-processor, to modify, display, and/or archive an NDT data object.
PRINT	Display all or part of an NDT data object
CONTRACT	Contract an NVT data object to an SVT data object (computational system vector) by extracting the independent DOFs
EXPAND	Expand a given vector to the NVT data object form, which includes values for specified and constrained freedoms
STOP	Exit the COP processor

Table 6.3-1 shows the order in which these commands would normally be employed in COP. Additionally, the CONSTRAIN sub-processor accepts the commands listed in Table 6.3-2.

Table 6.3-2 Sub-processor CONSTRAIN Command Summary

Command Name	Description
FREE	Declare freedom(s) to be independent, without constraints
ZERO	Single-Point-Constrain one or more DOFs to remain identically zero
NONZERO	Impose nonzero SPCnzs on one or more DOFs
MPC	Define a multi-point constraint relation
RESET	Reset a program-control parameter
SHOW	Display some or all of the NDT data object
DONE	Exit the CONSTRAIN sub-processor

The first step in executing COP is usually an invocation of the MODEL command, specifying a Complete Model Summary Table (CSM data object) that contains problem-size and other vital information for the model to be considered. This step is not required if the CSM data object to be used is that for the so-called zero-mesh case (where the mesh index in the dataset name for the CSM data object is zero); it is required for any other case.

The next step depends on what the user wants COP to do. To retrieve an existing NDT data object or construct a completely new one, and then to modify, archive or display that NDT data object, invoke the SELECT command to specify the starting NDT data object and the destination of the NDT data object that COP will produce, and then use the CONSTRAIN command (and its sub-commands) to define constraints and assign equation numbers. To contract (or expand) a system vector, bypass the SELECT command and use the CONTRACT (or EXPAND) command.

The PRINT, SEQUENCE, and RESET commands are optional. The PRINT command prints all or a selected part of a given NDT data object. The SEQUENCE identifies a Nodal-Ordering Table

(NOT data object) containing an {order} vector that defines the nodal sequence in which equation numbers are assigned to the active node points of the model. The RESET command specifies program-control parameters. The COP processor has three user-accessible control parameters that function as toggle (ON/OFF) switches to control assignment of equation numbers to all freedoms of the three basic types that COP recognizes: independent DOFs that are FREE (unconstrained); ZERO (trivially single-point-constrained, remaining forever zero); or NONZERO (single-point-constrained, with nonzero specified values). COP begins with these parameters set ON, so that equation numbers will be assigned for all such DOFs. To change those settings, the user must employ the RESET command before exiting the CONSTRAIN sub-processor.

The STOP command terminates execution of the COP processor, and must be the last command employed.

6.3.3 Command Glossary

6.3.3.1 MODEL Command

The first thing a COP user usually does is specify the Complete Model Summary Table (CSM data object) that contains the problem-size parameters and other vital information for the model to be treated. This is done with the MODEL command.

MODEL [*ldi_csm* [*dsn_csm*]]

The MODEL command opens the CSM data object stored in dataset *dsn_csm* on GAL library *ldi_csm* and extracts two problem-size parameters, NNODES (the maximum node point number for the model) and NDOFN (the maximum number of DOF that may be associated with each node).

The default value for *ldi_csm* is 1, and the default name for the Complete Model Summary (CSM data object) dataset is CSM.SUMMARY...0.

COP extracts the mesh index (*mesh*), and any other information needed to perform its function(s), from that CSM data object. The MODEL command is optional when the required CSM data object is identified by the default values described above; it is required for any other situation.

The MODEL keyword may be abbreviated to two characters.

6.3.3.2 SELECT Command

The SELECT command specifies a new or old (existing) NDT data object to initialize COP. It also specifies where the NDT data object produced by COP is to be archived. The syntax of the SELECT command is:

```
SELECT { NEW | OLD [ ldi_old [ cons [ mesh ] ] ] } ++
DOFDAT [ ldi_ndt [ icons [ imesh ] ] ]
```

where each keyword is defined below.

Keyword	Description
NEW	Indicates that a new NDT data object is to be constructed from scratch (using size and other information from the CSM data object identified in a previously-used MODEL command or from a default CSM data object if no MODEL command has been processed)
OLD	Indicates that an existing NDT data object is to be retrieved from GAL library *ldi_old*; the *cons* and *mesh* parameters (with default values of 1 and 0, respectively) designate the constraint case and the mesh index for the existing NDT data object
DOFDAT	Indicates that the NDT data object that COP produces is to be archived on GAL library *ldi_ndt*, in dataset NODAL.DOF..*icons.mesh*; the *icons* and *imesh* parameters default to *cons* and *mesh*, respectively

Given the NEW keyword, COP retrieves the NNODES and NDOFN parameters (and DOF type information) from the CSM data object identified in a previous MODEL command (or from the default CSM data object that COP uses if a MODEL command was not given) and constructs an initial NDT data object from scratch, giving each node the same number and types of DOF and setting the constraint status of each DOF to FREE (not constrained).

Given the OLD keyword (and optionally the *cons* and *mesh* parameters), COP attempts to retrieve an existing NDT data object from the indicated GAL library, and uses that Nodal DOF Table as the initial version, to be modified, displayed, and/or archived via the CONSTRAIN command (described below), or displayed via the PRINT command. If the indicated NDT data object is not found, COP prints an appropriate error message and terminates.

The SELECT keyword may be abbreviated to three characters. The NEW, OLD, and DOFDAT keywords may be abbreviated to one character.

6.3.3.3 SEQUENCE Command

The SEQUENCE command specifies an existing Nodal Order Table (NOT data object), which contains an {order} vector that defines the nodal sequence in which equation numbers are to be assigned to the active node points of the model. The syntax of the SEQUENCE command is:

```
SEQUENCE [ ldi_seq [ dsn_seq ] ]
```

Given the SEQUENCE command, COP opens the NOT data object stored in dataset *dsn_seq* on GAL library *ldi_seq*, and extracts the {order} vector from it. COP uses this {order} vector to assign an equation number to each DOF that is entitled to have an equation number, at each active node point of the model, when that operation is performed (prior to displaying the NDT data object and/ or exiting the CONSTRAIN sub-processor).

The default values of the *ldi_seq* and *dsn_seq* parameters on the SEQUENCE command are 1 and NODAL.ORDER...*mesh*, respectively, the mesh parameter being that which COP has extracted from the CSM data object specified via the MODEL command, or from the default CSM data object that COP attempts to use if the MODEL command was not used. The SEQUENCE command is optional. If it is not used, COP generates and uses a default {order} vector that gives sequence number 1 to the lowest-numbered active node, 2 to the next-lowest-numbered active node, ..., and so on to the highest-numbered active node.

The SEQUENCE keyword may be abbreviated to three characters.

6.3.3.4 RESET Command

The RESET command may be employed to reset a processor-control parameter. The syntax of the RESET command is:

```
RESET    [ FREE = {YES | NO} ]  [ ++
         [ NONZERO = {YES | NO} ]  [ ++
         [ ZERO = {YES | NO} ]
```

COP currently has three user-accessible program-control parameters, which control whether or not DOF with FREE, NONZERO, or ZERO constraint states are entitled to have equation numbers assigned to them when that operation is performed. COP is initialized with each of these switches in its ON (YES) position, so that each FREE, NONZERO, and ZERO constraint-status freedom is to be given an equation number. To suppress the assignment of an equation number to each freedom of any given type, use the RESET command to set the control parameter for that freedom type to its OFF (NO) value. This might be done for NONZERO and ZERO freedoms, for example, to assemble a system matrix with none of those freedoms present.

The RESET command keyword may be abbreviated to one character, and each key text word may be abbreviated to two characters.

6.3.3.5 DOF_SUPPRESS Command

The DOF_SUPPRESS command is used to modify a given NDT data object to set the constraint status of each DOF that is to be suppressed (single-point constrained to be zero) automatically. This is accomplished by identifying a DOF-suppression table (which must be an NDT data object), and using the constraint status information therein to superimpose the ZERO SPC pattern in the DOF-

suppression table onto a designated (input/output) NDT data object. The syntax for the DOF_SUPPRESS command is:

DOF_SUPPRESS INPUT = *ldi_inp inp_nam* [DOFDAT = *ldi cons mesh*]

where the two keywords are described below.

Keyword	Description
INPUT	Identifies dataset *inp_nam* on GAL library *ldi_inp* as the DOF-suppression table (NDT data object) that contains constraint status information to be used to modify the designated input/output Nodal DOF Table (NDT data object)
DOFDAT	Indicates that GAL library *ldi* contains the NDT data object to be modified; the *cons* and *mesh* parameters here indicate the constraint and mesh cases for the NDT data object to be used

The DOFDAT clause is optional on this command. If it is not included, the NDT data object identified in the previously-used SELECT command will be modified.

Given this command, COP retrieves the constraint status information for each active node in the NDT data object specified by the DOFDAT clause (or by the SELECT command, if the DOFDAT clause is omitted). COP also retrieves the constraint status information for the same node from the given DOF-suppression table. Each independent (non-multi-point-constrained) DOF for that node in the input/output NDT data object that has been SPCd to ZERO in the DOF-suppression table is then SPCd to ZERO in the input/output NDT data object. The DOF_SUPPRESS command only modifies the constraint status information in the input/output NDT data object. It does not assign equation numbers to DOF that are entitled to have them. That must be accomplished via the CONSTRAIN command, described below.

The DOF_SUPPRESS command and its two keywords may be abbreviated to one character.

6.3.3.6 CONSTRAIN Command

The CONSTRAIN command transfers the user into the CONSTRAIN sub-processor, which recognizes a set of sub-commands that facilitate the modification, display, and archiving of an NDT data object. The syntax of the CONSTRAIN command is very simple.

CONSTRAIN

The following sub-commands are recognized and processed by the CONSTRAIN sub-processor:

```
[ FREE      { NOD = i [ j [ nn ] ] }+  { DOF=typ1 [ typ2 [ ... ] ] }+ ]
[ ZERO      { NOD = i [ j [ nn ] ] }+  { DOF=typ1 [ typ2 [ ... ] ] }+ ]
[ NONZERO   { NOD = i [ j [ nn ] ] }+  { DOF=typ1 [ typ2 [ ... ] ] }+ ]
[ MPC       { ldi_mpc dsn_mpc | node dtype Nr αr ;
              nod1  typ1  C1
                :          Nr specifications
            ( nodNr  typNr  CNr }
[ RESET     [ FREE = {YES | NO} ]  [ ++
            [ NONZERO = {YES | NO} ]  [ ++
            [ ZERO = {YES | NO} ] ]
[ SHOW      [ n1 [ n2 ] ] ]
  DONE
```

6.3.3.6.1 *FREE Sub-command*

The FREE sub-command is used to declare that one or more freedoms at each of one or more node points is FREE (i.e., the freedoms in question are independent DOFs that are not subject to any constraints). The syntax of the FREE sub-command is:

```
FREE  { NOD=i  [ j [ nn ] ] }+  { DOF=typ1  [ typ2 [ ... ] ] }+
```

in which at least one NOD and at least one DOF clause must appear. The NOD and DOF clauses tell the CONSTRAIN sub-processor which degrees of freedom are to be "typed" through this command. Each NOD clause adds one or more nodes to a node-point list, and each DOF clause adds one or more directions to a direction list. The CON sub-processor uses these lists to set the type of freedoms in the direction list at each node point in the node list.

The *i* parameter is required in any given NOD clause, but *j* is optional and *nn* is second-order optional. If *j* is absent, only *i* goes into the node-point list; if *j* is present (but *nn* is not), nodes *i* through *j* (incrementing by plus or minus one, as appropriate) are added to the list; if *j* and *nn* are both present, nodes to be added to the list are determined by a FORTRAN-like loop of the form

```
do 10 k = i, j, nn

   NODE = k

10 continue
```

The node numbers thus specified must all must fall in the range $1 \leq \text{NODE} \leq \text{NNODES}$.

The same procedure is used for the construction of the direction list. This list is quite restricted: it must not be longer than the maximum number of freedoms NDOFN that can be accommodated at

any given node point (6, currently), and values in the list must be valid DOF type indicators for the problem at hand. With the current implementation of COP, the valid type indicators are *D1*, *D2*, *D3*, *Theta1*, *Theta2*, and *Theta3*, which represent translations in the *x*, *y*, and *z*-directions and rotations about the *x*, *y*, *and z*-axes.

The FREE keyword may be abbreviated to one character, but the NOD and DOF keywords must not be abbreviated.

6.3.3.6.2 ZERO Sub-command

The ZERO sub-command is used to declare that one or more freedoms at each of one or more node points is a ZERO-type freedom (i.e., the freedoms in question are independent DOFs that are constrained to be identically zero). The syntax of the ZERO command is:

ZERO { NOD=*i* [*j* [*nn*]] }$^+$ { DOF=typ_1 [typ_2 ,...]] }$^+$

where the meanings of the parameters following the ZERO keyword are the same as for the FREE command. The ZERO keyword may be abbreviated to one character, but the NOD and DOF keywords must not be abbreviated.

6.3.3.6.3 NONZERO Sub-command

The NONZERO sub-command is used to declare that one or more freedoms at each of one or more node points is a NONZERO-type freedom (i.e., the DOFs in question are independent DOFs that are constrained to be prescribed values that generally are nonzero). The syntax of the NONZERO command is

NONZERO { NOD=*i* [*j* [*nn*]] }$^+$ { DOF=typ_1 [typ_2 [...]] }$^+$

where the meanings of the parameters following the NONZERO keyword are the same as for the FREE command. Values are assigned to these freedoms via Processor AUS (see section 6.2). The NONZERO sub-command keyword may be abbreviated to one character, but the NOD and DOF keywords must not be abbreviated.

6.3.3.6.4 MPC Sub-command

COP gives the user the opportunity to specify that one or more of the freedoms for a given problem are linearly dependent upon the values of other freedoms, and to remove the dependent freedom(s) from the equation system for the analysis by using appropriate multi-point constraint relations where appropriate. This is facilitated by the MPC command, the syntax of which is

MPC *node dtype* N_r α_r

This command may be used to specify that the dtype degree of freedom at node point *node* is a linearly dependent MPC-type freedom and is to be eliminated from the equation system. The following multi-point constraint relation expresses the dependent freedom u_d in terms of the values of N_r independent freedoms $\{u_i\}$ and an (optional) intercept constant, α_r

$$u_d = \sum_{i=1}^{N_r} \left[C_i \times u_i \right] + \alpha_r$$

The $\{u_i\}$ are the N_r independent DOFs, and N_r, $\{C_i\}$, and α_r are constants.

The N_r independent DOFs and their associated weights must be specified via the N_r command addenda, which have the following syntax:

nod_k typ_k C_k

Each of the N_r independent freedoms is identified through its node and DOF-type specifications, nod_k and typ_k, respectively. A separate MPC command is required for each dependent freedom to be eliminated.

The MPC keyword may be abbreviated to one character if desired.

6.3.3.6.5 RESET Sub-command

The RESET sub-command here is exactly the same as described in Section 6.3.3.4. It may be exercised as many times as necessary in the CONSTRAIN subprocessor or in the COP main processor.

6.3.3.6.6 SHOW Sub-command

The SHOW sub-command displays the current NDT data object, while still under the control of the CONSTRAIN sub-processor. Information displayed includes the rectangular DOF-type and constraint status tables, and the rectangular DOF pointers table, which contains equation numbers for DOF that are entitled to have them, and pointers for constrained freedoms. The syntax for the SHOW command is

SHOW [n_1 [n_2]]

where the n_1 and n_2 parameters may be used to specify the first and last node numbers for which this information is desired. If n_1 is omitted, the entire NDT data object will be displayed. If n_1 is specified, but n_2 is omitted, information for node n_1 will be displayed. If n_2 is also specified, COP will display NDT data object information for nodes n_1 through n_2, inclusive.

The SHOW command may be abbreviated to one character, but at least two characters are recommended to prevent user confusion with the STOP command.

6.3.3.6.7 DONE Sub-command

The DONE sub-command tells the CONSTRAIN sub-processor that all relevant information has been defined for the current Nodal DOF Table (NDT data object). The syntax for the DONE sub-command is:

DONE

When the DONE command is issued, the CONSTRAIN sub-processor uses the information it has been given (including the default or nodal sequencing {order} vector) to assign an equation number to each freedom that is entitled to one (as discussed above) and to assign other appropriate pointer values to other freedoms. The finished NDT data object is then stored on the output GAL library, as specified via the SELECT command. Control then returns to the COP processor's main program, where COP waits for more selection, creation, manipulation, vector-transformation, program-control, and/or termination instructions.

The DONE command may be abbreviated to one character.

6.3.3.7 PRINT Command

The optional PRINT command causes the immediate printout of the information in an NDT data object. The syntax for the PRINT command is:

PRINT [DOFDAT = *ldi_ndt cons mesh*] [SUBSET = *first* [*last*]]

If the DOFDAT clause is omitted, the NDT identified as the output data object in the previously-used SELECT command will be printed. The DOFDAT clause permits the user to print a specific NDT data object (the one on GAL library *ldi_ndt* for which the constraint case and mesh case indices are *cons* and *mesh*) whether or not the SELECT command has been used. In any event, the entire NDT data object will be displayed if the SUBSET clause is omitted. The optional SUBSET clause may be used to specify the range of node points for which information is to be displayed, the *first* and *last* parameters indicating the desired range. If *last* is omitted, information will only be displayed for node *first*.

The PRINT keyword may be abbreviated to one character.

6.3.3.8 CONTRACT Command

The CONTRACT command contracts a system vector from the nodally-oriented NVT-data-object form to the DOF-oriented SVT-data-object form, eliminating dependent- and undefined-DOF as and if necessary. The syntax for the CONTRACT command is:

```
CONTRACT    INPUT  = ldi_inp inp_nam [ istep ] ++
            OUTPUT = ldi_out out_nam [ jstep ] [ ++
            DOFDAT = ldi cons mesh ]
```

where the three keywords are described below.

Keyword	Description
INPUT	Identifies dataset *inp_nam* on GAL library *ldi_inp* as the input NVT data object from which vector number *istep* is to be retrieved and contracted to the SVT-data-object (computational-vector) form
OUTPUT	Specifies that the contracted vector is to be stored as the *jstep* vector in the SVT data object in dataset *out_nam* on GAL library *ldi_out*
DOFDAT	Indicates that GAL library *ldi* contains the NDT data object to be used for the vector-transformation operation to be performed; the *cons* and *mesh* parameters here indicate the constraint and mesh cases for the NDT data object to be used.

The *istep* and *jstep* parameters default to 1 if they are not specified; and *jstep* defaults to *istep* if the former is given but the latter is omitted. The DOFDAT clause is optional on this command. If it is not included, the NDT data object identified in the SELECT command will be used.

The CONTRACT command may be abbreviated to four characters.

6.3.3.9 EXPAND Command

The EXPAND command produces an NVT data object by expanding a given input vector (which may be in the NVT-data-object or the SVT-data-object format) so that values corresponding to eliminated (dependent) DOF are reinstated using the multi-point constraint information in the specified NDT data object. The syntax of the EXPAND command is:

```
EXPAND      [ / { DOFVEC | NODVEC } ]  ++
            INPUT  = ldi_inp inp_nam [ istep ] ++
            OUTPUT = ldi_out out_nam istep ] [ ++
            VALUES = ldi_val val_nam [ scale ] ] [ ++
            DOFDAT = [ ldi cons mesh ] ]
```

The (optional) qualifier on this command may be used to specify that the vector to be expanded is in the SVT- or the NVT-data-object format. An SVT-data-object contains a computational vector with independent DOF only), while the NVT-data-object format accommodates a nodally-partitioned rectangular matrix. If no qualifier is given, COP assumes that the input vector is in the SVT-data-object format. The keywords and input parameters on this command are:

Keyword	Description
INPUT	Indicates that the vector to be expanded is stored in dataset *inp_nam* on GAL library *ldi_inp*, and that it is the *istep*th vector in this dataset (if the dataset contains an SVT data object)
OUTPUT	Indicates that the expanded, NVT data object resulting from this expansion operation is to be stored in dataset *out_nam* on GAL library *ldi_out*
VALUES	Optionally indicates that specified values needed for the expansion operation are to be obtained from the NVT data object stored in dataset *val_nam* on GAL library *ldi_val*; these values are to be multiplied by the scale factor *scale* before insertion into the expanded vector
DOFDAT	Optionally indicates that GAL library *ldi* contains the NDT data object to be used for the vector-transformation operation to be performed; the *cons* and *mesh* parameters here indicate the constraint and mesh cases for that NDT data object

The *istep* parameter defaults to 1 if it is not specified, and *scale* defaults to 1.0. The VALUES and DOFDAT clauses in this command are optional. If the DOFDAT clause is omitted, the NDT data object identified in the previously-used SELECT command will be used.

The EXPAND command may be abbreviated to three characters. The two qualifiers (DOFVEC and NODVEC) may be abbreviated to one character.

6.3.3.10 STOP Command

The STOP command should be the final instruction from the user to COP. The syntax of this command is:

```
STOP
```

The STOP command closes all active libraries, passes further-action instructions on to the next COMET-AR processor to be executed (if any), and terminates COP. COP also terminates with a RUN command (in the COMET-AR environment).

The STOP keyword may not be abbreviated!

6.3.4 Database Input/Output Summary

6.3.4.1 Input Datasets

A summary of input datasets for processor COP is given in Table 6.3-3.

Table 6.3-3 Processor COP Input Datasets

Dataset	Class	Description
CSM.SUMMARY...*mesh*	CSM	Model summary dataset. (Conditional)

Table 6.3-3 Processor COP Input Datasets

Dataset	Class	Description
NODAL.DOF.*cons..mesh*	NDT	Nodal DOF Table to initialize COP (Conditional)
Nodal.Order...*mesh*	NOT	Nodal Ordering Table (Optional)
Nodal.vecname.*step..mesh*	NVT	Nodal Vector Table(s) (Optional)
System.vecname...*mesh*	SVT	System Vector Table(s) (Optional)

The first two datasets in the above table are conditionally required. The model summary dataset is not required if the model-summary information to be used is stored on active GAL library 1 in dataset CSM.SUMMARY...0. It is required if that is not the case. The name of (*cons* and *mesh* parameters for) an existing Nodal DOF Table must be specified if one is used to initialize the NDT data object to be constructed by COP. The NOT-type input dataset in the above table is required only when the SEQUENCE command is used. This dataset must contain nodal-sequencing information (specifying the order in which nodes are to be assigned equation numbers for active, independent degrees of freedom). The name of this dataset is not hard-wired into COP and can be anything the user wishes. If a specific dataset is not designated (with the *ldi_seq* and *dsn_seq* parameters on the SEQUENCE command), COP uses the default name NODAL.ORDER...*mesh* (where the *mesh* index has been determined from the MODEL command or via default procedures used by the COP processor).

The NVT-type input dataset in the above table is required only when the CONTRACT command is used to convert an NVT data object to the SVT data object form, and/or when the EXPAND command is used to convert an NVT data object that does not contain corrected values of dependent degrees of freedom to one that does, and/or when the VALUES clause is used on the EXPAND command. The names of these datasets are not hard-wired into COP and may be anything that the user wishes.

The SVT-type input dataset in Table 6.3-3 is required only when the DOFVEC qualifier is used on the EXPAND command, indicating that the system vector to be expanded (given values for dependent degrees of freedom) is stored as an SVT data object.

6.3.4.2 Output Datasets

A summary of output datasets for processor COP is given in Table 6.3-4.

Table 6.3-4 Processor COP Output Datasets

Dataset	Class	Description
NODAL.DOF.*cons..mesh*	NDT	Nodal DOF Table (Optional)
Nodal.vecname.*step..mesh*	NVT	Nodal Vector Table (Optional)
System.vecname...*mesh*	SVT	System Vector Table (Optional)

Processor COP produces an output NDT data object for each invocation of the CONSTRAIN command. Each output Nodal DOF Table reflects constraint-state and other information (if any) specified by the user via CONSTRAIN sub-commands and/or nodal-ordering information specified via the (optional) SEQUENCE command. Processor COP produces an output NVT data object whenever the EXPAND command is used to convert an input system vector to an output system vector for which some of the degrees of freedom may have specified values (from a designated dataset) and others may be determined (for dependent freedoms) via multi-point constraint relations that are embedded within the designated Nodal DOF Table. The name of this dataset is not hard-wired into COP and may be anything that the user wishes. Processor COP produces an output SVT data object when the CONTRACT command is used to convert an input system vector from nodal to computational format. The name of this dataset is not hard-wired into COP and may be anything the user wishes.

6.3.5 Limitations

The current implementation of COP has two very important limitations. First, COP is currently a main-memory (an in-core) processor. All of the information required to generate an NDT data object, including the DOF type, state, and pointer tables, must simultaneously fit within the available core space, along with a nodal-ordering vector and any other information that may be required. When vector-transformation operations are requested, the required NDT and/or SVT data objects must also fit within the available memory space. Second, COP only understands NDT data objects for which every node point has the same number and types of degrees of freedom.

6.3.6 Error Messages

Processor COP produces more than a hundred self-explanatory error messages. Forty of these messages originate within the COP processor, with the remainder originating within the CSM*, NDT*, NOT*, NVT*, and SVT* utilities that COP uses for HDB-object management tasks.

COP responds to an unknown (probably misspelled) command keyword with a message of the form:

```
Unknown command = ...

COP recognizes the following commands:

...
```

with control returning to the appropriate command post for further (corrected) input. COP is not that user-friendly with most errors. COP responds to most error situations by printing a status message that is constructed at the point where the error is detected. COP terminates after attempting to close out and clean up any open GAL libraries that may be in use.

6.3.7 Examples and Usage Guidelines

Remember the two limitations discussed above, and refrain from using the current implementation of COP for large problems. The following examples are intended to show some (but not all) ways in which COP might be used within the current COMET-AR framework.

6.3.7.1 Example 1

COP might be used to form and display a new NDT data object for an unconstrained model constructed in the usual manner, with COMET-AR model-definition processors that produced (among other things) model summary information stored in dataset CSM.SUMMARY...7 on GAL library 3. The COP input for doing this is simple.

```
MODEL       = 3 CSM.SUMMARY...7          . Identify the CSM object
SELECT      = NEW DOFDAT = 3 1 7         . Build the new NDT object
CONSTRAIN                                . ->CONSTRAIN sub-processor
SHOW                                     . Display the NDT object
DONE                                     . Exit CONSTRAIN
STOP                                     . Exit COP
```

6.3.7.2 Example 2

For illustrative purposes, let us say that the model from the preceding example must be constrained to prohibit motion at nodes 1 through 6, and to impose a multipoint constraint on the Y translational DOF at node 100 to eliminate that DOF by relating it to the corresponding DOF at nodes 200 and 300. The COP input for doing that might look like the following.

```
MODEL = 3 CSM.SUMMARY...7                . Identify the CSM object
SELECT OLD= 3 1 7 DOFDAT = 3 2 7         . Build the new NDT object
CONSTRAIN                                . ->CONSTRAIN sub-processor
   ZERO NOD = 1,6 ++                     . Nodes 1 -> 6
     DOF = D1,D2,D3,Theta1,Theta2,Theta3 . 6 freedoms at each node
   MPC = 100 D2 2 0.0                    . node, dtype, Nr, and a
         200 D2 0.5                      . nod1 type coef
         300 D2 0.5                      . nod2 type coef
   DONE                                  . Exit CONSTRAIN
STOP                                     . Exit COP
```

6.3.7.3 Example 3

For the same model, suppose a DOFVEC-formatted vector was generated by a computational-vector-oriented processor, and is to be expanded to an NVT data object that includes all single- and multi-point constrained values:

```
MODEL = 3 CSM.SUMMARY...7                    . Identify the CSM object
SELECT OLD = 3 1 7 DOFDAT = 3 2 7            . Build new NDT object
EXPAND/DOFVEC ++                             . Expansion operation:
   INPUT  = 3 EXISTING.DOFVEC ++             . Input SVT object
   OUTPUT = 3 DESTINATION.NODVEC ++          . Output NVT object
   VALUES = 3 SPECIFYVALS.NODVEC ++          . NVT object
   DOFDAT = 3 2 7                            . NDT object
STOP                                         . Exit COP
```

where the prescribed values are in the NVT data object in dataset SPECIFYVALS.NODVEC on GAL library 3.

6.3.8 References

[1] Stanley, G. M. and Swenson, L. HDB: *Object-Oriented Database Utilities for COMET-AR,* NASA Computational Structural Mechanics (CSM) Contract Report, August 1992.

[2] Felippa, Carlos A., *The Computational Structural Mechanics Testbed Architecture: Volume II—Directives*, NASA CR-178385, February 1989.

6.4 Processor GCP (Generic Constitutive Processor)

See Chapter 8, *Constitutive Processors*, which covers all related functions, including pre-processing, solution, and post-processing phases of COMET-AR analysis.

6.5 Processor GEP (Generic Element Processor)

See Chapter 7, *Element Processors*, which covers all related functions, including pre-processing, solution, and post-processing phases of COMET-AR analysis.

6.6 Processor PST (COMET-AR_to_PATRAN)

See Chapter 14, *Post-Processors*, for a full description of Processor PST pre-processing (COMET-AR_to_PATRAN) and post-processing (PATRAN_to_COMET-AR, etc.) capabilities.

6.7 Processor REDO (Dataset Reformatter)

6.7.1 General Description

Processor REDO reformats various datasets created by processors TAB and AUS, converting them from the old (COMET) data structures to the new data objects required by COMET-AR. The TAB datasets that currently need reformatting by REDO are: the model summary (JDF1.BTAB); nodal coordinates (JLOC.BTAB); and nodal transformations (QJJT.BTAB) datasets. These are converted to CSM.SUMMARY (CSM format), NODAL.COORDINATE (NCT format) and NODAL.TRANSFORMATION (NTT format) datasets. The AUS datasets that currently need reformatting are the applied nodal force and applied nodal displacement datasets, both of which are converted from the old SYSVEC format to the new NVT format (in datasets named NODAL.*Vector*) by REDO. Processor REDO also has a copy function for nodal vectors.

6.7.2 Command Summary

Processor REDO follows standard COMET-AR command interface protocol. A summary of valid commands is given in Table 6.7-1.

Table 6.7-1 Processor REDO Command Summary

Command Name	Function
CSM	Converts model summary dataset from JDF1 format to CSM.SUMMARY (CSM) format.
NCT	Converts nodal coordinate dataset from JLOC format to NODAL.COORDINATE (NCT) format.
NTT	Converts nodal transformation dataset from QJJT format to NODAL.TRANSFORMATION (NTT) format.
NVT	Converts nodal vector datasets from SYSVEC format to NODAL.*Vector* (NVT) format.
NVT /COPY	Copies NODAL.Vector (NVT) datasets and creates new dataset with different name.

6.7.3 Command Definitions

6.7.3.1 The CSM Command

The CSM command creates a COMET-AR model summary (CSM) dataset, CSM.SUMMARY...*mesh*, from a TAB-generated JDF1.BTAB dataset.

Command Format:

CSM [*JDF1_ldi,*] *JDF1_dsname* [*CSM_ldi,*] [*CSM_dsname*]

where

Keyword	Description
JDF1_ldi	Logical device index of database input file containing JDF1 dataset. (Default: 1)
JDF1_dsname	Name of JDF1 dataset (e.g., JDF1.BTAB.*).
CSM_ldi	Logical device index of database input file containing CSM dataset. (Default: 1)
CSM_dsname	Name of CSM dataset. (Default: CSM.SUMMARY...*mesh*)

Only nodal summary parameters are created in the CSM summary dataset via this command. All element summary parameters must be added subsequently by the generic element processor (ES*i*).

6.7.3.2 The NCT Command

The NCT command creates a COMET-AR nodal coordinate (NCT) dataset, NODAL.COORDINATE...*mesh*, from a TAB-generated JLOC.BTAB dataset.

Command Format:

NCT [*JLOC_ldi,*] *JLOC_dsname* [*NCT_ldi,*] *NCT_dsname*

where

Keyword	Description
JLOC_ldi	Logical device index of database input file containing JLOC dataset. (Default: 1)
JLOC_dsname	Name of JLOC dataset (e.g., JLOC.BTAB.*).
NCT_ldi	Logical device index of database input file containing NCT dataset. (Default: 1)
NCT_dsname	Name of NCT dataset. (Default: NODAL.COORDINATE...*mesh*)

It is assumed that a CSM summary dataset, named CSM.SUMMARY...*mesh*, is on the data library connected to the same *ldi* as the NCT dataset (i.e., *NCT_ldi*), and that the mesh index appearing in the CSM dataset name is the same as that appearing in *NCT_dsname*.

6.7.3.3 The NTT Command

The NTT command creates a COMET-AR nodal transformation (NTT) dataset, NODAL.TRANSFORMATION...*mesh*, from a TAB-generated QJJT.BTAB dataset.

Command Format:

NTT [*QJJT_ldi,*] *QJJT_dsname* [*NTT_ldi,*] *NTT_dsname*

where

Keyword	Description
QJJT_ldi	Logical device index of database input file containing QJJT dataset. (Default: 1)
QJJT_dsname	Name of QJJT dataset (e.g., QJJT.BTAB.*).
NTT_ldi	Logical device index of database input file containing NTT dataset. (Default: 1)
NTT_dsname	Name of NTT dataset. (Default: NODAL.TRANSFORMATION...*mesh*)

It is assumed that a CSM summary dataset, named CSM.SUMMARY...*mesh*, is on the data library connected to the same *ldi* as the NTT dataset (i.e., *NTT_ldi*), and that the mesh index appearing in the CSM dataset name is the same as that appearing in *NTT_dsname*.

6.7.3.4 The NVT Command

The NVT command creates a COMET-AR nodal vector (NVT) dataset, NODAL.*Vector...mesh*, from an AUS-generated SYSVEC dataset. A typical application of this command is to convert applied force and displacement datasets from SYSVEC to NVT data formats.

Command Format:

NVT [*SYSVEC_ldi,*] *SYSVEC_dsname* [*NVT_ldi,*] *NVT_dsname*

where

Keyword	Description
SYSVEC_ldi	Logical device index of database input file containing SYSVEC dataset. (Default: 1)
SYSVEC_dsname	Name of SYSVEC dataset. (Example: APPL.FORC.1.1)
NVT_ldi	Logical device index of database input file containing NVT dataset. (Default: 1)
NVT_dsname	Name of NVT dataset. (Example: NODAL.SPEC_FORCE.1..*mesh*)

It is assumed that a CSM summary dataset, named CSM.SUMMARY...*mesh*, is on the data library connected to the same *ldi* as the NVT dataset (i.e., *NVT_ldi*), and that the *mesh* index appearing in the CSM dataset name is the same as that appearing in *NVT_dsname*.

6.7.3.5 The NVT/COPY Command

Use of the /COPY qualifier with the NVT command indicates that both the input and output datasets are in the NVT data format.

Command Format:

NVT/COPY [*NVT1_ldi,*] *NVT1_dsname* [*NVT2_ldi,*] *NVT2_dsname*

where

Keyword	Description
NVT1_ldi	Logical device index of database file containing input NVT dataset. (Default: 1)
NVT1_dsname	Name of input NVT dataset. (Example: NODAL.SPEC_FORCE.1..*mesh1*)
NVT2_ldi	Logical device index of database file containing output NVT dataset. (Default: 1)
NVT2_dsname	Name of output NVT dataset. (Example: NODAL.SPEC_FORCE.1..*mesh2*)

It is assumed that a corresponding CSM summary dataset exists for each of the NVT datasets, on the corresponding *ldi*, and with corresponding *mesh* index.

6.7.4 Database Input/Output

6.7.4.1 Input Datasets

A summary of input datasets required by Processor REDO is given in Table 6.7-2.

Table 6.7-2 Processor REDO Input Datasets

Dataset	Class	Contents
APPL.FORC.* APPL_DISP.*	SYS-VEC	AUS-generated applied force and/or displacement datasets (NVT command)
CSM.SUMMARY...*mesh*	CSM	REDO-generated model summary dataset (NCT, NTT, NVT, and NVT/COPY commands).
JDF1.BTAB.*	JDF1	TAB-generated model summary dataset (CSM command).
JLOC.BTAB.*	JLOC	TAB-generated nodal coordinate dataset (NCT command).
NODAL.*Vector...mesh*	NVT	Nodal vector dataset (NVT/COPY command).
QJJT.BTAB.*	QJJT	TAB-generated nodal transformation dataset (NTT command).

6.7.4.2 Output Datasets

A summary output datasets created by Processor REDO is given in Table 6.7-3.

Table 6.7-3 Processor REDO Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset (CSM command).
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate dataset (NCT command).
NODAL.TRANSFORMATION...*mesh*	JLOC	Nodal transformation dataset (NTT command).

Table 6.7-3 Processor REDO Input Datasets

Dataset	Class	Contents
NODAL.*Vector...mesh*	NVT	Nodal vector dataset (NVT/COPY command).

6.7.5 Limitations

6.7.5.1 Limitation 1: Element Data

Processor REDO does not reformat element summary data via the CSM command. Such data must be added to the CSM.SUMMARY dataset by the normal operation of the generic element processor, (i.e., the REDO CSM command must be invoked before performing element definition).

6.7.6 Error Messages

All error messages generated by processor REDO are related to missing datasets, as:

"Unable to access dataset XXX"

where XXX represents the name of the missing dataset.

6.7.7 Examples and Usage Guidelines

6.7.7.1 Example 1: Reformatting Summary, Nodal Coords, and Nodal Transformations

```
run REDO
            CSM     JDF1.BTAB.*     CSM.SUMMARY
            NCT     JLOC.BTAB.*     NODAL.COORDINATE
            NTT     QJJT.BTAB.*     NODAL.TRANSFORMATION
stop
```

In the above example, the model summary, nodal coordinate, and nodal transformation datasets are converted from the TAB-generated (COMET) formats into new COMET-AR formatted datasets. The original datasets are left unmodified.

6.7.7.2 Example 2: Reformatting Applied Nodal Forces and Displacements

```
run REDO
                NVT    APPL.FORC.1      NODAL.SPEC_FORCE.1
                NVT    APPL.MOTI.1      NODAL.SPEC_DISP.1
stop
```

In the above example, applied nodal force and displacement datasets are converted from AUS-generated (COMET) SYSVEC formats into new (COMET-AR) NVT formatted datasets. The original datasets are left unmodified.

6.7.8 References

None.

6.8 Processor RENO (Node Renumbering)

6.8.1 General Description

Processor RENO performs a simple geometrical nodal renumbering with special provisions for the AR environment.

RENO employs a simple geometric ordering technique: an ordering direction vector is computed as a unit vector from the lowest point in the structure to the highest one via:

$$\boldsymbol{u} = \frac{maxX - minX}{\|maxX - minX\|}$$

where

$$minX_i = \min_{n \in Nnodes} \left(X_i^n\right)$$

$$maxX_i = \max_{n \in Nnodes} \left(X_i^n\right)$$

Once the ordering direction is computed, RENO defines a plane perpendicular to the ordering direction and moves this plane from the minimum point to the maximum point. As each nodal point passes through that plane it is renumbered.

In practice, RENO computes a sorting weight for each node: $w = \boldsymbol{X} \cdot \boldsymbol{u}$ and employs a quicksort algorithm to sort the nodes according to their assigned weights. The resulting sorted order of the nodes is used as the renumbering table.

Processor RENO is typically invoked by a high-level AR control procedure, such as AR_CONTROL (via procedure L_STATIC_1), in an adaptive refinement iteration loop.

6.8.2 Command Summary

Processor RENO follows standard COMET-AR command interface protocol. A summary of valid commands is given in Table 6.8-1.

Table 6.8-1 Processor RENO Command Summary

Command Name	Function	Default Value
SET MESH	Specifies mesh number for renumbering	0
SET CONSTRAINT_SET	Specifies constraint-set number	1
SET LDI	Specifies logical device index of computational database	1

Table 6.8-1 Processor RENO Command Summary (Continued)

Command Name	Function	Default Value
RENO/*qualifier*	Renumber the specified mesh	

6.8.3 Command Definitions

6.8.3.1 SET CONSTRAINT_SET Command

This command defines the constraint set number associated with the element and nodal data. This number should appear as the second cycle number in names of all element and nodal datasets.

Command syntax:

SET CONSTRAINT_SET = *conset*

where

Parameter	Description
conset	Constraint set number (default value: 1)

6.8.3.2 SET LDI Command

This command defines the logical device index for the computational database.

Command syntax:

SET LDI = *ldi*

where

Parameter	Description
ldi	Logical device index (default value: 1)

6.8.3.3 SET MESH Command

This command defines the mesh number for the mesh to be renumbered.

Command syntax:

SET MESH = *mesh*

where

Parameter	Description
mesh	Mesh to be renumbered (default value: 0)

6.8.3.4 RENO Command

This is the "go" command for processor RENO. It causes RENO to generate the renumbering record for the mesh.

Command syntax:

RENO/*qualifier*

where

Parameter	Description
qualifier	Renumbering option (default value: FULL): FULL - renumber all nodes in the mesh PARTIAL - renumber only new nodes of the specified mesh; nodes of previous meshes retain their original numbering.

6.8.4 Database Input/Output

6.8.4.1 Input Datasets

A summary of input datasets required by Processor RENO is given below in Table 6.8-2.

Table 6.8-2 Processor RENO Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate dataset
NODAL.DOF..*conset.mesh*	NDT	Nodal DOF dataset
EltNam.DEFINITION...*mesh*	EDT	Element definition dataset

6.8.4.2 Output Datasets

A summary of output datasets created by Processor RENO is given in Table 6.8-3.

Table 6.8-3 Processor RENO Output Datasets

Dataset	Class	Contents
NODAL.ORDER...*mesh**	NOT	Nodal ordering dataset

*—created dataset

6.8.5 Limitations

6.8.5.1 Sub-Optimal Ordering

RENO was designed to be a quick solution with special capabilities required by the AR environment (e.g., partial renumbering). The simple algorithm employed by RENO for renumbering the nodes is inferior to the more advanced algorithms employed by the RSEQ processor. Use of RENO should be limited to simple geometry problems and the *s*-refinement method only, since RSEQ is not capable of partial renumbering. For *s*-refinement, RSEQ may be employed for renumbering the initial mesh.

6.8.6 Error Messages

RENO contains extensive error checking. Most of the error messages printed by RENO are self-explanatory and aim to help the user correct mistakes. Some of the errors may occur at code levels below RENO (e.g., HDB, DB, GAL etc.) and RENO describes those errors to the best of its ability.

Table 6.8-4 summarizes the error messages related to user interface problems produced by RENO.

Table 6.8-4 Processor RENO Error Messages

Index	Error Message	Cause	Recommended User Action
1	Unknown SET *variable name* encountered in RENO.	RENO user interface cover encountered an unrecognized SET variable name.	Check the spelling of *variable name* in the CLIP procedure.
2	Unknown *command* encountered in RENO.	RENO user interface cover encountered an unrecognized COMMAND.	Check the spelling of the *command* in the CLIP procedure.

Table 6.8-4 Processor RENO Error Messages (Continued)

Index	Error Message	Cause	Recommended User Action
3	*Old/new dataset name* could not be opened in *routine name*.	RENO could not open a certain dataset.	1. Check the execution log file; look for error produced by processors prior to RENO execution. 2. Try to verify the particular *dataset name* using the HDBprt processor. 3. Make sure that all required input datasets are present in the database file.
4	*Dataset name* could not be closed in *routine name*.	RENO could not close a certain dataset.	1. Check the execution log file; look for errors previously produced by processor RENO. 2. Verify that RENO is the ONLY PROCESSOR accessing the database file (is ARGx being used in the same directory?).
5	*Dataset name* access problem encountered in *routine name* or could not get/put/add/update *attribute name* to *dataset name* in *routine name*.	RENO could not get/put an attribute from/to the dataset name table.	Verify that the particular *dataset name* contain attributes required by RENO.

In addition to the above generic messages, RENO will print any relevant information regarding the problem such as element data, nodal data and geometry information to assist the user in correcting the error. A full trace-back printout of error messages will follow the first message, and RENO will attempt to terminate its execution as cleanly as possible (closing opened datasets, releasing memory allocations, etc.).

6.8.7 Examples and Usage Guidelines

6.8.7.1 Example 1: Basic Operation

```
*run RENO
                    SET MESH                    = 1
                    RENO
stop
```

In the above example, all nodal points in mesh 1 are being renumbered.

6.8.7.2 Example 2: Partial Renumbering

```
*run RENO
                SET MESH                    = 2
                RENO/PARTIAL
stop
```

In the above example, only new nodes generated during the adaptive refinement from mesh 1 to mesh 2 are being renumbered. Nodes that existed in mesh 1 retain their previous ordering.

6.8.8 References

None.

6.9 Processor RSEQ (Node Renumbering)

6.9.1 General Description

Processor RSEQ constructs a joint elimination sequence by any one of four methods: Nested Dissection (fill minimizer), Minimum Degree (fill minimizer), Reverse Cuthill-McKee (profile minimizer), and Gibbs-Poole-Stockmeyer (bandwidth minimizer). The first three methods (Nested Dissection (N/D), Minimum Degree (M/D), and Reverse Cuthill-McKee (RCM)), were all taken from Reference [1], while the Gibbs-Poole-Stockmeyer (GPS) algorithm was taken from the BANDIT program documentation supplied in Reference [2].

For large problems, significant savings in CPU times can usually be realized by employing one of the four joint elimination sequences. Each of the available methods work well for some, usually different, problems.

6.9.2 Command Summary

Processor RSEQ follows standard COMET-AR command interface protocol. A summary of RSEQ commands is given in Table 6.9-1.

Table 6.9-1 Processor RSEQ Command Summary

Command Name	Function	Default
SET/LIB	Specifies data library with element data	1
SET/MAXCON	Specifies maximum number of joints connected to a single joint	0
SET/METHOD	Specifies method of nodal reordering	0
SET/MESH	Specifies adaptive refinement mesh number	0
SET/CONSTRAINT	Specifies constraint set for identifying input dataset	1

6.9.3 Command Definitions

6.9.3.1 SET/LIB Command

This command sets the GAL Library where input and output datasets for the resequencing reside.

Command Format:

SET /LIB = *ldi*

where

Keyword	Description
ldi	GAL Library containing element definitions and destination library for the nodal ordering dataset. (Default: 1)

6.9.3.2 SET/MAXCON Command

This command sets the maximum number of joints connected to any one joint. Using the default value of 0 results in the automatic computation of the value of *maxcon* based on the assumption of 2D built-up structures. While *maxcon* must be set to at least the maximum connectivity of any one joint, it may have a value larger than this maximum.

Command Format:

SET /MAXCON = *maxcon*

where

Keyword	Description
maxcon	Maximum number of joints connected to a single joint. (Default: 0)

6.9.3.3 SET/METHOD Command

This command selects the method for nodal resequencing of the model.

Command Format:

SET /METHOD = *method*

where

Keyword	Description
method	Method of determining joint elimination sequence. (Default: 0) 0 — Nested Dissection (Fill minimizer) 1 — Minimum Degree (Fill minimizer) 2 — Reverse Cuthill-McKee (profile minimizer) 3 — Gibbs-Poole-Stockmeyer (bandwidth minimizer)

6.9.3.4 SET/MESH Command

This command selects the adaptive refinement mesh number to use for nodal resequencing.

Command Format:

SET /MESH = *mesh*

where:

Keyword	Description
mesh	The adaptive refinement mesh number for the resequencing. (Default: 0)

6.9.3.5 SET/CONSTRAINT Command

This command selects the constraint set number to be used for nodal resequencing.

Command Format

SET /CONSTRAINT = *constraint_set*

where

Keyword	Description
constraint_set	Sets the constraint set for the resequencing. (Default: 1)

6.9.4 Database Input/Output

6.9.4.1 Input Datasets

A summary of input datasets required by Processor RSEQ is given in Table 6.9-2.

Table 6.9-2 Processor RSEQ Input Datasets

Dataset	Type	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset
NODAL.DOF..*constraint_set.mesh*	NDT	Nodal DOF table. Contains information about nodal freedoms and multipoint constraints.

6.9.4.2 Output Datasets

A summary of output datasets created by Processor RSEQ is given in Table 6.9-3.

Table 6.9-3 Processor RSEQ Output Datasets

Dataset/Attribute	Type	Contents
NODAL.ORDER...*mesh*	NOT	Nodal ordering table containing a list of node numbers arranged in the sequence to be used for equation numbering.

6.9.5 Limitations

In general, the Gibbs-Poole-Stockmeyer method requires the largest memory working space. The Reverse Cuthill-McKee and Nested Dissection methods each have the same minimum space requirement. Table 6.9-4 lists the exact memory requirements of each of the methods.

Table 6.9-4 Memory Requirements for RSEQ Methods

Method	Space Required
Nested Dissection (N/D)	$\Omega + 3J$
Minimum Degree (M/D)	$\Omega + 7J$
Reverse Cuthill-McKee (RCM)	$\Omega + 3J$
Gibbs-Poole-Stockmeyer (GPS)	$\Omega + 9J + 2$

where

J = Number of Joints

M = Maximum Connectivity

Ω = (Number of element types)+(Record Length)+17+2(J+1)+J*M

There may be difficulties, especially with very large models, in processing a new joint sequence with ASM (because of the space requirements of ASM). In some cases it may be impossible to run the new elimination sequence as the new connectivity exceeds the limit on available data space.

The algorithms used in RSEQ currently do not take into account the multipoint constraints generated by h_c-, or h_s-refinement. Consequently, this omission generally leads to non-optimal equation ordering.

6.9.6 Error Messages

Checks are made within RSEQ to ensure that enough work space is available. Before processing begins, if there is not enough room to form either the adjacency arrays or the new numbering, a message will be printed to the output file and execution will stop. The message will contain both the space required and the space available.

In addition to checks on the amount of space available, a check is made on the amount of space that has been allocated by the user through the SET /MAXCON command. If the value of MAXCON is too small, execution will terminate with a message indicating the first joint at which MAXCON was exceeded.

6.9.7 Examples and Usage Guidelines

```
RUN RSEQ
        SET /MESH = 1
        SET /METHOD = 3
STOP
```

In the above example RSEQ will renumber equations for mesh 1 using the Gibbs-Poole-Stockmeyer algorithm.

6.9.8 References

[1] George, Alan and W-H Liu, J., *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.

[2] Everstine, G. C., *The BANDIT Computer Program for the Reduction of Matrix Bandwidth for NASTRAN*, NSRDC Report 3872, March 1972.

6.10 Processor TAB (Tabulation of Nodal Coordinates)

6.10.1 General Description

Processor TAB contains an array of subprocessors which are used by COMET-AR to generate tables of node (also called joint) locations and reference frames for the structure. TAB may be used to either (1) create new datasets, or (2) update existing datasets by replacing individual entries in them. TAB subprocessors create datasets of which the first part of the name is the same as the name of the subprocessor, the second part is BTAB, and the third and fourth parts are unique to each subprocessor. These datasets are subsequently translated into High Level Database (HDB) objects by processor REDO, described in Section 6.7.

6.10.2 Command Summary

Processor TAB follows the SPAR command syntax as described in Reference [1]. A summary of valid commands is given in Table 6.10-1.

Table 6.10-1 Processor TAB Command Summary

Command Name	Function
START	Model size declaration
JLOC	Joint (node) location subprocessor
ALTREF	Alternative reference frame subprocessor
JREF	Joint (node) reference frame subprocessor
FORMAT	Sets the format parameter for a subprocessor
UPDATE	Sets the update mode on/off
NREF	Sets the NREF parameter for a subprocessor
MOD	Adds a constant to subsequent node numbers
ONLINE	Controls amount of processor printout

6.10.3 Command Definitions

6.10.3.1 START Command

When beginning a new problem, the first data line following RUN TAB must be the START command, the syntax for which is:

START *j*

where

Parameter	Description
j	Total number of joints in the structure.

It is not harmful to have some unused joints (i.e. joints connected to no elements) for convenience in interpreting the output. This should not be carried to extremes, however, since it wastes memory.

6.10.3.2 JLOC Command Subprocessor

The JLOC subprocessor produces a table containing the position coordinates of the joints (i.e., nodes). The data sequence on input lines is as follows:

JLOC

$k, x_1^A, x_2^A, x_3^A, [x_1^B, x_2^B, x_3^B, ni, [ijump, [nj]]]$

if nj is given, a second input line must appear,

$[jjump, x_1^C, x_2^C, x_3^C, x_1^D, x_2^D, x_3^D]$

where the input parameters are as described below.

Parameter	Description
k	Joint number
x_1^A, x_2^A, x_3^A	Coordinates of point A
x_1^B, x_2^B, x_3^B	Coordinates of point B
ni	Number of equally spaced points between points A and B
$ijump$	Joint number increment between points A and B (Default = 1)
nj	Number of equally spaced points between points A and C
$jjump$	Joint number increment between points A and C (Default = 1)
x_1^C, x_2^C, x_3^C	Coordinates of point C
x_1^D, x_2^D, x_3^D	Coordinates of point D

There are three possible interpretations of the above:

1) If only k, x_1^A, x_2^A, and x_3^A are given, the x^A's are interpreted as the coordinates of joint k.

2) If k, x_1^A, ..., x_3^B, ni, and $ijump$ are given, the x^A's and x^B's are coordinates of points A and B, which terminate a string of ni equally-spaced joints.

3) If nj is given, a linearly interpolated two-dimensional mesh of ni by nj joints is defined.

Although the output table generated by JLOC is in rectangular coordinates relative to the global frame, coordinate data appearing on the input lines may be in either rectangular or cylindrical coordinates and may be relative to any frame already defined via ALTREF. The associated command interpretations for the NREF and FORMAT commands within the JLOC subprocessor are summarized below.

Command	Meaning for JLOC subprocessor
NREF = *n*	Coordinate data on subsequent lines are relative to frame *n* (until another NREF command is encountered).
FORMAT = 2	Subsequent data are in cylindrical coordinates, relative either to frame 1 (global) or to any other frame selected by an NREF command. The convention is shown below.
FORMAT=1	Switch back to rectangular coordinates.

Switching among frames and between rectangular and cylindrical coordinates is unrestricted.

If cylindrical coordinates are used in connection with mesh generation, interpolation is performed before transformation to rectangular coordinates so that regular meshes on circles, cylinders, and cones are readily generated.

6.10.3.3 ALTREF Command Subprocessor

In addition to the global reference frame, the analyst may find it convenient to define alternate reference frames. These frames have several uses, including the following:

1) Joint locations may be defined in any frame the analyst finds most convenient (see the JLOC command).

2) Joint reference frame orientations may be defined using the alternate frame (see the JREF command).

The command format for the ALTREF subprocessor is:

ALTREF

$k, i_1, a_1, i_2, a_2, i_3, a_3, x_1, x_2, x_3$

where

Parameter	Description
k	Integer identifying the alternative reference frame.
i_1, i_2, i_3	Axis numbers about which rotations are to be performed. Valid values are 1, 2, and 3, in any order.
a_1, a_2, a_3	Angles, in degrees, of rotation about the axes specified by the i_1, i_2, i_3 parameters above.
x_1, x_2, x_3	Position coordinates, relative to the global frame, of the origin of frame k. The x's need not be given if only the orientation of frame k is of significance, which often is the case.

The associated command interpretations for the FORMAT commands within the ALTREF subprocessor are summarized below.

Command	Meaning for JLOC subprocessor
FORMAT = 1	The processing sequence is: (1) rotate the local frame a_1 degrees about local axis i_1, then (2) from the new position, rotate the local frame a_2 degrees about axis i_2, then (3) from the resulting position, rotate the local frame a_3 degrees about axis i_3. (Default)
FORMAT = 2	The *i*'s and *a*'s indicate rotation of the global frame relative to frame *k*.

Each frame is uniquely identified by a positive integer. The following three alternate reference frames are generated automatically upon entering TAB.

1) Global frame: x_{alt} coincident with x_{global}

2) x_{alt} coincident with y_{global}

3) x_{alt} coincident with z_{global}

While the global frame is always frame 1, frames 2 and 3 may be overwritten by the analyst. In this case, a message warning that a predefined reference frame is being overwritten will be written to output and execution will continue.

6.10.3.4 JREF Command Subprocessor

A unique right-hand rectangular reference frame is associated with each joint (i.e., node). Through JREF the analyst may designate the orientation of any joint reference frame. All joint reference frames not defined in JREF are, by default, parallel to frame 1 (global). The orientations of these frames is of considerable significance, since joint motion components and mechanical loads applied at joints are defined relative to the joint reference frames.

The command format for the JREF subprocessor is:

NREF = *n*
joint_data

where

Parameter	Description
n	Number of the alternative reference frame to be used for the joint. If *n* is negative then the interpretation of the joint reference frame is as follows. The 3-axis of the joint frame is parallel to the three axis of frame *n*. The 1-axis of the joint frame is perpendicular to the 3-axis of frame *n*. See the ALTREF command for details about alternative reference frames.
joint_data	Set of joints in loop-limit format for which the alternate reference applies.

6.10.3.5 NREF Command

This command is used to set the NREF parameter to apply to subsequent input for the current subprocessor. The command format for the NREF command is:

NREF = *n*

where,

Parameter	Description
n	A positive integer parameter used in subprocessors JLOC and JREF to specify the reference frame which applies to data on subsequent lines. This parameter is automatically reset to its default value of 1 at the beginning of execution of a new subprocessor.

6.10.3.6 FORMAT Command

This command is used to set the FORMAT parameter to apply to subsequent input for the current subprocessor. The command format for the FORMAT command is:

FORMAT = *j*

where,

Parameter	Description
j	An integer parameter specifying one of the admissable formats for the current subprocessor. Details are given in discussions of the individual subprocessors. This parameter is automatically reset to its default value of 1 at the beginning of execution of a new subprocessor.

6.10.3.7 UPDATE Command

This command is used to enable or disable modification of existing datasets. The command format for the UPDATE command is:

UPDATE = *n*

where

Parameter	Description
n	To enter the "update"' mode of operation, the command UPDATE = 1 is used. To leave the update mode, the command UPDATE = 0 is used. (Default = 0)

The UPDATE command should immediately precede subprocessor execution commands. When operating in the update mode, the output dataset produced in the current execution is identical to

that produced in the preceding execution, except for entries defined by the command input of the current execution. As an example, suppose the location of joint 1742 is found to be in error. The JLOC dataset could be repaired by the following command sequence.

```
UPDATE = 1
JLOC
   1742, 947.62, 1841.9 23.487
UPDATE = 0
```

6.10.3.8 MOD Command

This command is used to set the MOD parameter to apply to subsequent input for the current subprocessor. The command format for the MOD command is:

```
MOD = m
```

where,

Parameter	Description
m	A positive integer added to joint (i.e., node) numbers specified on subsequent input lines by the JLOC or JREF commands. This parameter is automatically reset to the default value of 0 when any of these commands is issued.

6.10.3.9 ONLINE Command

This command is used to control the amount of printout written by processor TAB. The command format for the ONLINE command is:

```
ONLINE = n
```

where,

Parameter	Description
n	An integer that may be set to: 0 for minimum printout, 1 for normal printout or 2 for maximum printout.

The ONLINE command may be used more than once within a given TAB execution.

6.10.3.10 Input Datasets

If the START command is not the first input line to TAB, the JDF1.BTAB.1.8 dataset must be present in the computational database. In UPDATE mode, any datasets modified must be present.

6.10.3.11 Output Datasets

A summary of output datasets created by Processor TAB is given in Table 6.10-2.

Table 6.10-2 Processor TAB Output Datasets

Dataset	Contents
JDF1.BTAB.1.8	Model size (Number of joints)
ALTR.BTAB.2.4	Alternate reference frame data
JLOC.BTAB.2.5	Rectangular coordinates of each joint
JREF.BTAB.2.6	Orientation of joint reference frame(s)
QJJT.BTAB.2.19	Joint reference frame orientation

6.10.4 Limitations

TAB is a processor within the COMET-AR macroprocessor. As such, there is a blank common limit which is installation dependent. TAB will notify the user if the memory required for processing the commands is insufficient, in which case the user can increase the blank common of the executable.

6.10.5 Error Messages

Error messages may be printed by TAB subprocessors; messages denoted as FATAL will cause termination of the TAB processor execution. These errors are summarized in Table 6.10-3.

Table 6.10-3 Error Messages For Processor TAB

Error Message and Meaning	Subprocessor
******COORDINATE INPUT ERROR. I, J, JLOCAL, JOINT = xxxx Joint number out of range 1 to number of joints	JLOC
CORE INADEQUATE TO FORM QJ(3,3,JT). AVAIL, REQ= x	JLOC, ALTREF
ERROR, K=1 NOT ALLOWED The global frame is always 1; therefore, alternate reference frames must start with 2.	ALTREF
***ERRORS IN INPUT PREVENT CALCULATION OF QJ(3,3,JT) One of the following datasets is marked in error as a result of input data errors: ALTR.BTAB.2.4 JLOC.BTAB.2.5 JREF.BTAB.2.6	JLOC, ALTREF
FATAL ERROR. NERR, N=INCD n Too many data items on input line (>40)	All
FATAL ERROR. NERR, N=JT n Invalid number of joints (n < 2) specified on START command.	START

Table 6.10-3 Error Messages For Processor TAB (Continued)

Error Message and Meaning	Subprocessor
FATAL ERROR. NERR, N=KORE n Common block size is too small by n words; must be at least 13 times the number of joints	START
FATAL ERROR. NERR, N=NREF n Invalid reference frame specified on NREF command; entry not defined in ALTR.BTAB.2.4	NREF
*** FATAL INPUT ERROR. NEGATIVE INDEX OR INSUFFICIENT CORE. NAME= xxxx	JREF
***ILLEGAL JOINT, x	JREF
INPUT ERROR, JO= x Input data line is out of order; numeric data expected.	JLOC
UNRECOGNIZED DATA SKIPPED. INPUT DATA ERROR, RECORD x	All
*** WARNING. ERRORS IN SOURCE DATA A disabled dataset was encountered.	All
*** WARNING. x SETS OF DATA MISSING, ARRAY xxxx This is an informational message only. The parameter x may have a negative value; for example, this would occur if a joint location was defined more than once, which is allowed within a single execution of subprocessor JLOC.	All

6.10.6 Examples and Usage Guidelines

When beginning a new problem, the first data line should be a START command and the computational data library should be empty. If the UPDATE command is used, it should appear immediately before a subprocessor execution command. The NREF command may be used repeatedly within the input stream of any subprocessor. At the beginning of execution of a new subprocessor, NREF is reset to the default value.

6.10.7 References

[1] Whetstone, W. D., "Computer Analysis of Large Linear Frames," *J. Struct. Div.*, ASCE, Vol. 95, No. ST11, Proc. Paper 6897, November 1969, pp. 2401-2417.

[2] Whetstone, W. D., Yen, C. L., and Jones, C. E., *SPAR Structural Analysis System Reference Manual, System Level 13A: Volume 2: Theory*, NASA CR 158970-2, December 1978.

6.11 Processor NODAL

6.11.1 General Description

Processor NODAL defines nodal quantities during COMET-AR model definition. Presently, the only nodal quantities that can be defined with processor NODAL are lumped nodal masses to be added to the system mass matrix.

6.11.2 Command Summary

Processor NODAL follows standard COMET-AR command interface protocol. A summary of valid commands is given in Table 6.11-1.

Table 6.11-1 Processor NODAL Command Summary

Command Name	Function
DEFINE MASS	Converts model summary dataset from JDF1 format to CSM.SUMMARY (CSM) format.
END_DEFINE	Converts nodal coordinate dataset from JLOC format to NODAL.COORDINATE (NCT) format.

6.11.3 Command Definitions

6.11.3.1 The DEFINE MASS Command

The DEFINE MASS command is used to define lumped nodal masses at selected nodes and degrees of freedom (DOFs). The command has the following syntax.

Command Syntax:

```
DEFINE MASS [ = mass_dataset ]

        MASS = mass_value  NODES = nodseq  DOFS = dofseq
          :
          :

END_DEFINE
```

where

Parameter	Description
mass_dataset	Name of the dataset in which to output the nodal masses. This dataset is of type NVT (Nodal Vector Table). Default: NODAL.MASS (recommended!)
mass_value	Value of lumped mass to be added to set of nodes and DOFs specified by NODES and DOFS keywords.
nodseq	List of nodes, in "do-loop" format, i.e., *first_node, last_node, node_increment.*
dofseq	List of nodal DOF numbers, in arbitrary order, e.g., *i1, i2, i3,* ...; where *in* is an integer less than the maximum number of DOFs per nodes (typically 6).

The MASS phrase may be repeated as often as necessary to define all lumped masses to be added to the system. Any nodes/DOFs not mentioned in a MASS phrase are assumed to have zero added mass. The specification of a node/DOF more than once does not lead to an accumulation of the mass values; but rather the last mass value specified will override.

6.11.3.2 The END_DEFINE Command

The END_DEFINE command is used to terminate any of the DEFINE commands (e.g., DEFINE MASS), and must be entered to ensure that the DEFINE command is processed completely.

Command Syntax:

```
END_DEFINE
```

6.11.4 Database Input/Output

6.11.4.1 Input Datasets

A summary of input datasets required by Processor NODAL is given in Table 6.11-2.

Table 6.11-2 Processor REDO Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset.
NODAL.DOFS.*conset..mesh*	NDT	Nodal DOF dataset

6.11.4.2 Output Datasets

A summary output dataset created by Processor NODAL is given in Table 6.11-3.

Table 6.11-3 Processor NODAL Input Dataset

Dataset	Class	Contents
NODAL.MASS...*mesh*	NVT	Lumped nodal masses to be added to the system mass matrix, defined via the DEFINE MASS command.

6.11.5 Limitations

6.11.5.1 Limitation 1: Body Forces

Lumped added masses at specified nodal DOFs are applied only to the system mass matrix; they are not used during the computation of body forces, which are assumed (at this time) to be element-based quantities, determined from distributed load data (e.g., force per unit mass).

6.11.6 Error Messages

All error messages generated by processor NODAL are related to missing datasets, and have the form:

"Unable to access dataset XXX"

where XXX represents the name of the missing dataset.

6.11.7 Examples and Usage Guidelines

6.11.7.1 Example 1: Adding a Lumped Mass to All Translational DOFs at Selected Nodes

```
run NODAL
                DEFINE MASS = NODAL.MASS
                        MASS = 2.25        NODES=1,10  DOFS=1,2,3
                        MASS =3.25         NODES=11,20  DOFS=1,2,3
                END_DEFINE
stop
```

In this example, a mass of 2.25 is added to all translational DOFs at nodes 1 through 10, and a mass of 3.25 is added to all translational DOFs at nodes 11 through 20.

6.11.7.2 Example 2: Adding a Lumped Mass to All Rotational DOFs at All Nodes

```
run NODAL
                    DEFINE MASS = NODAL.MASS
                            MASS = 2.25          DOFS=4,5,6
                    END_DEFINE
stop
```

In this example, an lumped mass of 2.25 units is added to all rotational DOFs at all nodes in the model; the absence of the NODES phrase implies (by default) all nodes. The same is true of the DOFS phrase. If absent, it is assumed that the MASS specification applies to all DOFS at the specified list of nodes. If both NODES and DOFS phrases are missing, the mass value is applied to all DOFs at all nodes.

6.11.8 References

None.

6.12 Processor GM2AM (Geometric to Analysis Model)

6.12.1 General Description

Processor GM2AM generates an initial analysis mesh (mesh 0) by refining a geometry model of higher-order (i.e., 16-node shell) elements according to user-specified requirements.

The reference geometry mesh used by the GM2AM processor is a standard COMET-AR database defined using either a CLIP model procedure or a PATRAN 16-node element mesh translated into a COMET-AR database via PST. In either case, GM2AM requires the reference geometry mesh to consist of quadrilateral 16-node shell elements (topologically equivalent to those generated by element processors ES1p or ES7p with cubic polynomial order, i.e., p=3).

GM2AM generates the solid-model interface (SMI) hooks between the initial analysis mesh and the reference geometry model so that the refinement processor may use the 16-node element reference geometry mesh during all subsequent adaptive refinement operations.

> Processor GM2AM is typically executed automatically for you by invoking the high-level command procedure, GM2AM. Refer to Section 2.13 for use of procedure GM2AM.

GM2AM is a two-phase processor: an INITIALIZE phase followed by a REFINEMENT phase. In the INITIALIZE phase GM2AM scans the user input and extracts the element names requested for the analysis model. GM2AM uses these element names to generate an initialization procedure called *init_elt* which must be invoked after the INITIALIZE phase (this is performed automatically by procedure GM2AM). The purpose of the *init_elt* procedure is to initialize the element definition and interpolation tables (i.e., EDT and EIT) in the analysis database by executing the DEFINE ELEMENTS command using the appropriate element processors. A second purpose of the INITIALIZE phase is to copy the Generic Constitutive Processor (GCP) records from the geometry model to the analysis model database.

In the REFINEMENT phase GM2AM generates the initial analysis mesh by refining the reference geometry model according to user-specified requirements.

Due to the two-phase execution of GM2AM, the user will find it convenient to use an "add" file containing the requirements for generating the initial analysis mesh; "gm2am.add" is the default name used by the generic GM2AM procedure. This "add" file may be used for both phases of GM2AM execution without modification as described in Section 2.13.

The following sections describe the commands recognized by the GM2AM processor.

6.12.2 Command Summary

Processor GM2AM follows standard COMET-AR command interface protocol. A summary of GM2AM commands is given below in Table 6.12-1.

Table 6.12-1 Processor GM2AM Command Summary

Command Name	Function	Default Value
INITIALIZE	Go command for the INITIALIZE phase	
PROCESS_GMELTS	Specifies the list of geometry model elements to be processed	none
REFINE	Go command for the REFINEMENT phase	
SET CONSTRAINT_SET	Specifies constraint-set number	1
SET ELEMENT_NAME	Specifies the element name to be generated in the analysis mesh by the following PROCESS_GMELTS lists	none
SET GROUP_ID	Specifies group ID to be attached to each analysis mesh element generated in each geometry element specified by the following PROCESS_GMELTS lists	Geometry Element ID
SET LDI_AM	Specifies logical device index of analysis model database	2
SET LDI_GM	Specifies logical device index of geometry model database	1
SET LOAD_SET	Specifies load-set number	1
SET NEL_X	Specifies number of elements to be generated along the horizontal shell coordinate direction in each geometry element specified by the following PROCESS_GMELTS lists	1
SET NEL_Y	Specifies number of elements to be generated along the vertical shell coordinate direction in each geometry element specified by the following PROCESS_GMELTS lists	1
SET P	Specifies the polynomial order of the element specified by the SET ELEMENT_NAME command	none
SET STEP	Specifies the load/time step number	0

6.12.3 Command Definitions

6.12.3.1 INITIALIZE Command

This is the "go" command for processor GM2AM initialization phase. It causes GM2AM to scan user refinement requirements (i.e., from geometric elements to analysis elements) and to collect all element names specified by the user. The collected list of element names is used by GM2AM to construct an element initialization procedure, "init_elt.clp", specific for the model at hand. (The element initialization procedure, "init_elt.clp", is used by the generic GM2AM procedure to

initialize the element definition and interpolation tables in the analysis database by executing the DEFINE_ELEMENT command in each of the appropriate element processors.)

The INITIALIZE phase of GM2AM must be executed prior to the REFINEMENT phase.

Command syntax:

INITIALIZE

6.12.3.2 PROCESS_GMELTS Command

This command specifies a list of geometry-model elements to be refined, in order to create an initial analysis model. Each 16-node geometry element within the list will be refined into NEL_X × NEL_Y *element_name* elements and all the relevant datasets for these elements will be added to the analysis database.

Command syntax:

PROCESS_GMELTS = *list*

where

Parameter	Description
list	List of 16-node geometry model elements to be refined: = 0—process all 16-node geometry elements = *Elt_ID*—process a single 16-node geometry element = *first,last*—process all 16-node geometry elements in the range *first* through *last* = *first,last,step*—process all 16-node geometry elements in the range *first* through *last* in increment *step* (default value: none)

6.12.3.3 REFINE Command

This is the "go" command for processor GM2AM's refinement phase. It causes GM2AM to scan user refinement requirements and to refine the reference geometry model according to the user's specifications.

The INITIALIZE phase of GM2AM must be executed prior to the REFINEMENT phase.

Command syntax:

```
REFINE
```

6.12.3.4 SET CONSTRAINT_SET Command

This command defines the constraint set number associated with the element and nodal data in both the geometry and the analysis models. This number appears as the second cycle number in the names of all element and nodal datasets.

Command syntax:

```
SET CONSTRAINT_SET = conset
```

where

Parameter	Description
conset	Constraint set number (default value: 1)

6.12.3.5 SET ELEMENT_NAME Command

This command defines the name of the next element to be generated in the analysis model by the PROCESS_GMELTS commands that follow it. This name appears as the first component in the names of all element datasets.

Command syntax:

```
SET ELEMENT_NAME = elt_name
```

where

Parameter	Description
elt_name	Element name string of the form: *ProcName_EltType* where: *ProcName* — Element processor name *EltType* — Element type within the processor (default value: none)

6.12.3.6 SET GROUP_ID Command

This command the group ID for the next elements to be generated in the analysis model by the PROCESS_GMELTS commands that follow it.

Command syntax:

SET GROUP_ID = *group_id*

where

Parameter	Description
group_id	Element group ID number (default value: parent geometry element ID)

6.12.3.7 SET LDI_AM Command

This command defines the logical device index for the analysis database file.

Command syntax:

SET LDI_AM = *ldi_am*

where

Parameter	Description
ldi_am	Logical device index of the analysis model database (default value: 2)

Due to dataset naming conventions in COMET-AR the analysis and geometry datasets must be in different databases; thus, *ldi_am* ≠ *ldi_gm*.

6.12.3.8 SET LDI_GM Command

This command defines the logical device index for the geometry model database file.

Command syntax:

SET LDI_GM = *ldi_gm*

where

Parameter	Description
ldi_gm	Logical device index of the geometry model database (default value: 1)

Due to dataset naming conventions in COMET-AR the analysis and geometry datasets must be in different databases; thus, *ldi_am* ≠ *ldi_gm*.

6.12.3.9 SET LOAD_SET Command

This command defines the load set number associated with the element data in both the geometry and the analysis models. This number appears as the first cycle number in the names of all element load datasets.

Command syntax:

SET LOAD_SET = *ldset*

where

Parameter	Description
ldset	Load set number (default value: 1)

6.12.3.10 SET NEL_X Command

This command defines the user refinement requirements for the "ξ" shell element natural coordinate directions. Each 16-node geometry element specified by subsequent PROCESS_GMELTS commands will be refined to that number of elements along the ξ-direction of the geometry element.

Command syntax:

SET NEL_X = *nel_x*

where

Parameter	Description
nel_x	Number of elements along the shell element ξ direction (default value: 1)

User refinement requirements are restricted to compatible meshes; that is, neighboring 16-node geometry elements must have the same number of nodes generated along their common boundary.

6.12.3.11 SET NEL_Y Command

This command defines the user refinement requirement for the η shell-element natural coordinate direction. Each 16-node geometry element specified by subsequent PROCESS_GMELTS commands will be refined to that number of elements along the η-direction.

Command syntax:

SET NEL_Y = *nel_y*

where

Parameter	Description
nel_y	Number of elements along the η natural-coordinate direction (default value: 1)

User refinement requirements are restricted to compatible meshes; that is, neighboring 16-node geometry elements must have the same number of nodes generated along their common boundary

6.12.3.12 SET P Command

This command defines the polynomial order of the elements which will be generated in the analysis database by subsequent PROCESS_GMELTS commands. This value is mandatory and required for the INITIALIZE phase of GM2AM.

Command syntax:

SET P = *p*

where

Parameter	Description
p	Element polynomial order (default value: 0)

6.12.3.13 SET STEP Command

This command defines the solution step number associated with the element and nodal data in both the geometry and the analysis models. This number appears as the second cycle number in names of all element and nodal datasets.

Command syntax:

SET STEP = *step*

where

Parameter	Description
step	solution step number. (default value: 0)

6.12.4 Database Input/Output

6.12.4.1 Input Datasets

A summary of input datasets required by Processor GM2AM is given below in Table 6.12-2 for the geometry model database and in Table 6.12-3 for the analysis database. The datasets listed in Table 6.12-3 are actually output datasets during the INITIALIZE phase of GM2AM execution, and then become input datasets during the REFINE phase of GM2AM execution.

Table 6.12-2 Processor GM2AM Input Datasets (Geometry Model)

Dataset	Class	Contents
CSM.SUMMARY	CSM	Model summary dataset
NODAL.COORDINATE	NCT	Nodal coordinate dataset
NODAL.DOF..*conset*	NDT	Nodal DOF dataset
NODAL.TRANSFORMATION	NTT	Nodal transformation dataset
NODAL.SPEC_FORCE.*ldset*	NVT	Nodal specified force dataset
NODAL.SPEC_DISP.*ldset*	NVT	Nodal specified displacement dataset
EltNam.DEFINITION	EDT	Element definition dataset
EltNam.INTERPOLATION	EIT	Element interpolation dataset
EltNam.FABRICATION	EFT	Element fabrication dataset
EltNam.LOAD	ELT	Element loads datasets

Table 6.12-3 Processor GM2AM Input Datasets (Analysis Model)

Dataset	Class	Contents
CSM.SUMMARY	CSM	Model summary dataset
EltNam.DEFINITION	EDT	Element definition dataset
EltNam.INTERPOLATION	EIT	Element interpolation dataset

6.12.4.2 Output Datasets

A summary of output datasets created by Processor GM2AM is given below in Table 6.12-4.

Table 6.12-4 Processor GM2AM Output Datasets (Analysis Model)

Dataset	Class	Contents
CSM.SUMMARY	CSM	Model summary dataset
NODAL.COORDINATE*	NCT	Nodal coordinate dataset
NODAL.DOF.*conset**	NDT	Nodal DOF dataset
NODAL.TRANSFORMATION*	NTT	Nodal transformation dataset
NODAL.SPEC_FORCE.*ldset**	NVT	Nodal specified force dataset
NODAL.SPEC_DISP.*ldset*	NVT	Nodal specified displacement dataset
EltNam.DEFINITION	EDT	Element definition dataset
EltNam.GEOMETRY*	EGT	Element geometry (solid model links) dataset
EltNam.FABRICATION*	EFT	Element fabrication dataset
EltNam.LOAD*	ELT	Element loads datasets

*—created dataset

6.12.5 Limitations

6.12.5.1 Compatible Meshes

GM2AM is limited to compatible mesh generation. The user must ensure that neighboring 16-node geometry model elements will be requested to refine such that each will generate the same number of nodes along their common edges. GM2AM will abort its execution upon detection of a non-compatible refinement request.

The compatibility of the mesh is enforced only in terms of number of nodes along common edges. There is no restriction regarding the compatibility of the displacement field. For example, the user may refine one 16-node geometry element into a single 9-node shell element and its neighboring 16-node element into four 4-node shell elements, generating three nodes along their common edge. This is not recommended unless the user also plans to add some form of multi-point compatibility constraint.

6.12.5.2 16-Node Quadrilateral Geometry Elements

GM2AM currently recognizes only 16-node quadrilateral elements. Serendipity 12-node elements, triangular elements and 3D tri-cubic elements are not supported by GM2AM at this time. Triangular domains may be represented as collapsed quadrilateral 16-node elements (using the

standard convention in which all nodes along the third element edge are set to an identical node ID number).

6.12.5.3 Nodal Data Limitations

The user should be cautious in specifying discrete nodal data (such as nodal forces or lumped quantities) in the geometry model. The reason for this is that not all the nodes of the geometry model will be active in the analysis model (e.g., be used as an analysis element nodal point). Only the four 16-node geometry element corner nodes in the geometry model are guaranteed to be present and active in the analysis mesh. All other nodes in the geometry model will be active in the analysis model only if they are used as nodal points by at least one active analysis element.

For example, if a 16-node geometry element is requested to be refined into a single 4-node analysis shell element, then only the four corner nodes will be present in the analysis model. The other twelve nodes will not be active and their nodal information will not be part of the analysis. If, on the other hand, the same element is requested to be refined into 3x3 4-node analysis shell elements, all of the original 16 nodes will be active in the analysis model.

In general, concentrated nodal forces and masses are not recommended for AR.

6.12.5.4 Separate Database Files Limitations

The geometry model and the analysis model databases cannot be generated in the same file. This restriction is due to some dataset naming convention limitations within COMET-AR. The user must generate the geometry model database in a separate file and open both the geometry and analysis databases prior to executing any GM2AM commands.

6.12.6 Error Messages

GM2AM contains extensive error checking. Most of the error messages printed by GM2AM are self-explanatory and aim to help the user correct mistakes. Some of the errors may occur at code levels below GM2AM (e.g., HDB, DB, GAL, etc.), and GM2AM describes those errors to the best of its ability.

Table 6.12-5 summarizes the error messages produced by GM2AM that are related to user interface problems.

Table 6.12-5 Processor GM2AM Error Messages

Index	Error Message	Cause	Recommended User Action
1	Unknown SET *variable name* encountered in GM2AM.	GM2AM user interface cover encountered an unrecognized SET variable name.	Check spelling of *variable name* in CLIP procedure.
2	Unknown *command* encountered in GM2AM.	GM2AM user interface cover encountered an unrecognized command.	Check spelling of *command* in CLIP procedure.
3	*Old/new dataset name* could not be opened in *routine name.*	GM2AM could not open a certain dataset.	1. Check execution log file for error produced by processors prior to GM2AM execution. 2. Verify the particular dataset name using the HDBprt processor. 3. Make sure that all required input datasets are present in the database files.
4	*Dataset name* could not be closed in *routine name.*	GM2AM could not close a certain dataset.	1. Check the execution log file for errors previously produced by processor GM2AM. 2. Verify that GM2AM is the only processor accessing the database file. (Is ARGx being used in the same directory?)
5	*Dataset name* access problem encountered in routine name or could not get/put/add/update *attribute name* to *dataset name* in *routine name.*	GM2AM could not get/put an attribute from/to the dataset name table.	Verify that the particular dataset contains attributes required by GM2AM (e.g., NDT contains nontrivial data).
6	Unknown Geometry Entity ID encountered in SMShl*xx*, *entity type* ID = *entityID.*	The solid model interface shell routines in GM2AM could not locate a geometry model entity.	Verify that the elements specified by the PROCESS_GMELTS command are present in the geometry model.
7	Solid Model Interface Problem encountered in SMShl*xx.*	The solid model interface shell routines in GM2AM could not perform their current task.	Verify that the elements specified by the PROCESS_GMELTS command are present in the geometry model.
8	Convergence problem encountered in *xxx*Proj, could not locate projected point along *geometry entity type.*	The solid model interface shell routines in GM2AM could not project a new point into the boundaries of the corresponding geometry model entity.	1. Verify that the elements specified by the PROCESS_GMELTS command are present in the geometry model. 2. Verify that the topology of the 16-node geometry element being refined is correct by inspecting the geometry model using ARGx.
9	Inconsistent refinement request	The user specified refinement requests that yielded different number of nodes along a common 16-node geometry elements edge.	Verify that all refinement requests will yield the same number of nodes along common 16-node geometry element edges.

In addition to generic messages in Table 6.12-5, GM2AM will print any relevant information about the problem such as element data, nodal data, and geometry information, to assist you in correcting the error. A full trace-back printout of error messages will follow the first message, and GM2AM will attempt to terminate its execution as cleanly as possible (by closing opened datasets, releasing memory allocations, etc.).

6.12.7 Examples and Usage Guidelines

6.12.7.1 Example: INITIALIZE Phase

```
RUN GM2AM
              INITIALIZE
                              SET ELEMENT_NAME        = ES1_EX97
                              SET P                   = 2
                              SET NEL_X               = 3
                              SET NEL_Y               = 2
                              PROCESS_GMELTS          = 0
STOP
```

In this example, the INITIALIZE phase of GM2AM is performed. In this phase, GM2AM will generate the "init_elt_clp" procedure which will be used to initialize the 9-ANS element (EX97) in ES1 element processor tables. The actual initialization of the element datasets requires the following CLIP command directives.

```
*open 1 TEST.DBG

*open 2 TEST.MODEL.DBC

*add init_elt.clp

*copy 2 = 1, FABRICATIONS

*copy 2 = 1, MATL.*
```

The above directives are automatically invoked when procedure GM2AM is used to run processor GM2AM (see Section 2.13).

6.12.7.2 Example: REFINE Phase

```
RUN GM2AM

              SET LDI_GM      = 1

              SET LDI_AM      = 2

              REFINE


                              SET ELEMENT_NAME        = ES1_EX97

                              SET P                   = 2

                              SET NEL_X               = 3

                              SET NEL_Y               = 2

                              PROCESS_GMELTS          = 0

STOP
```

In this example, the REFINEMENT phase of GM2AM is performed. In this phase, GM2AM will refine each 16-node geometry element in the geometry database (logical device unit 1) into 3x2 9-node ANS elements in the analysis model (logical device unit 2).

The shaded commands in the above box give an example of the "gm2am.add" file, which is all that is required as command input when employing procedure GM2AM to perform the initial analysis model creation, rather than processor GM2AM.

6.12.8 References

[1] Stanley, G., Levit, I., Hurlbut, B., and Stehlin, B., *Adaptive Refinement Strategies for Shell Structures: Part 1: Preliminary Research*, NASA Contractor Report, 1991.

[2] Stanley, G., Levit, I., Hurlbut, B., Stehlin, B., Loden, W., and Swenson, L., *COMET–AR: Adaptive Refinement (AR) Manual*, NASA Contractor Report, May 1991.

7 Element Processors

7.1 Overview

In this chapter, the generic element processor (GEP) is described as well as various specific element-type processors that are based on the GEP architecture. The GEP is a standard processor template, or "shell", from which many individual element processors can be constructed. All the individual element processors share a common user interface (i.e., commands/functions) and a common database interface with the generic element processor. We refer to the generic element processor shell as ES (for Element/Structural), and name each of the individual (special-purpose) element processors ES*i*, where the *i* denotes a sequence number and/or alphanumeric string (e.g., ES1p or ES36, etc.). Each ES*i* processor performs all of the functions associated with elements of a particular type(s). This includes pre-processing functions (e.g., element connectivity and load definition), and solution functions (e.g., forming element force, stiffness, and stress arrays). Regarding constitutive functions (e.g., stress and constitutive matrix calculation), these are performed by the generic element processor via the generic constitutive processor (GCP), the solution portion of which is embedded in each element processor as a constitutive utility library.

A summary of currently available element processors is given in Table 7.1-1. Each processor is described in a separate section within this chapter. The ES section describes the generic commands and relevant database entities. The ES*i* sections then describe the specific element types available in processor ES*i*, and any element-type oriented command options and/or database parameters not covered in the generic (ES) section.

Table 7.1-1 Outline of Chapter 7: Element Processors

Section	Processor	Function
7.2	ES	Generic Element Processor (template for ES*i*)
7.3	ES1	SRI and ANS Shell Elements
7.4	ES5	STAGS Shell Element
7.5	ES6	STAGS Beam Element
7.6	ES1p	Variable-p Lagrange Quadrilateral Shell Elements
7.7	ES7p	ANS Shell Elts; Var. Order Quads
7.8	ES36	MIN3/6 Triangular Shell Elements

7.2 Processor ES (Generic Element Processor)

7.2.1 General Description

The generic element processor, or ES (for Element/Structural), provides a standard template with which many individual structural finite-element processors may be developed and coexist as independent modules in COMET-AR. Specific element processors built with the ES template all have names that begin with ES (e.g., ES1p, ES7p, ES36, etc.). Each of these ES*i* processors performs all operations for all element types implemented within the processor, including the definition of element connectivity and loads during pre-processing, the formation of element force and stiffness arrays during the primary solution phase, and the formation of strains and stresses during the post-processing phase of a structural analysis.

This section describes the standard user command and database interfaces employed by the generic element processor (ES) and shared by all individual ES*i* processors based on the ES template. All of the standard functions performed by element processors are described in this section. (For theoretical and developer documentation on the generic element processor, consult Reference [1].) A special subsection is also included to indicate which ES commands may be invoked via the convenient ES Utility Procedure, which automatically executes all element processor and types associated with a given model. For most analyses, users will not have to directly interact with the generic element (ES) processor or procedure except (perhaps) during model definition, via the DEFINE ELEMENTS and DEFINE LOADS commands. Otherwise, element functions are automatically exercised via solution procedures and their subordinate utility procedures (see Part II of this manual).

7.2.2 Command Classes

The generic element processor (ES) commands are partitioned into several classes. A summary of these command classes is given in Table 7.2-1. A separate subsection is then devoted to each class.

Table 7.2-1 Generic Element Processor (ES) Command Classes

Command Class	Function
DEFINE	Element definition commands used during pre-processing phase; includes element connectivity, loads, and other attributes.
INITIALIZE	Solution initialization command used just before solution phase.
FORM	Element formation commands used during solution phase; includes force, stiffness, mass, and stress formation.
RESET	Element parameter reset options. Some reset parameters are mandatory.

7.2.3 ES Processor DEFINE Commands

A summary of the DEFINE commands accessible via the generic element processor (ES) is given in Table 7.2-2.

Table 7.2-2 Summary of ES DEFINE Commands

DEFINE Command	Function
DEFINE ELEMENTS	Defines element connectivity; includes nodal connectivity, material (fabrication) pointers, and material reference frame. This command must be employed before any of the other commands in this section.
DEFINE LOADS	Defines element applied distributed loads; includes line loads, pressures, surface tractions, and body forces.
DEFINE FREEDOMS	Defines valid element nodal freedoms for subsequent automatic freedom suppression.
DEFINE NORMALS	Defines element nodal normal vectors (for plate and shell elements only).
DEFINE DRILL_FLAGS	Defines element nodal drilling stabilization flags (for plate and shell elements only)
DEFINE ATTRIBUTES	Defines general element attributes (e.g., temperature, moisture, etc.)

The DEFINE ELEMENTS and DEFINE LOADS commands are the most important commands for model definition. The other DEFINE commands are typically invoked automatically by various solution and utility procedures.

7.2.3.1 The DEFINE ELEMENTS Command

The DEFINE ELEMENTS command is used to define element connectivity—nodal, fabrication, and solid-model—for elements of a particular type. The element type name must be set via the RESET ELEMENT_TYPE command prior to issuing the DEFINE ELEMENTS command.

7.2.3.1.1 Command Syntax

The DEFINE ELEMENTS command is composed of a nested group of subcommands and qualifiers that allows the definition of all elements of a particular type. Element nodal connectivity is defined via a separate subcommand (ELEMENT=...) for each element, while other parameters such as element group numbers and fabrication association may be defined via a separate phrase that remains intact for all subsequent ELEMENT subcommands until a new definition of the phrase is encountered. The meta-language description of the DEFINE ELEMENTS command and its subcommands is as follows.

```
DEFINE ELEMENTS [ /SOLID_MODEL = solid_model_option ]  [ /p = polynomial_order ]
               [ GROUP = group_number ]
               [ FAB_ID = fabrication_id ]
               [ FAB_DIR = fabrication_direction ]
               [ FAB_ECC = fabrication_eccentricity ]
               [ SURFACE = solid_model_surface_id ]

               ELEMENT = e1    NODES = n1, n2, . . . nnen          [ LINES = l1, l2, ..., lnle ]
               ELEMENT = e2    NODES = n1, n2, . . . nnen          [ LINES = l1, l2, ..., lnle ]
                                  :
               ELEMENT = enel  NODES = n1, n2, . . . nnen          [ LINES = l1, l2, ..., lnle ]
END_DEFINE_ELEMENTS
```

7.2.3.1.2 The /SOLID_MODEL Qualifier

The /SOLID_MODEL qualifier may be used to pick the solid model interface (SMI) option. The format is:

```
/SOLID_MODEL = solid_model_option
```

where the available options are:

solid_model_option	**Description**
USER	A user-written solid model interface will be created (see Chapter 16, *Solid Model Interface*). Links to user-written subroutines will be provided via the SURFACE and LINE subcommands within the DEFINE ELEMENTS command.
DISCRETE (Default)	The initial finite-element model will suffice as the solid-model description. (SURFACE and LINE subcommands become irrelevant.)

7.2.3.1.3 The /P Qualifier

The /P qualifier may be used to pick the solid model interface (SMI) option. The format is:

```
/P = polynomial_order
```

where *polynomial_order* is the element polynomial_order (e.g., 1, 2, 3, ...). This qualifier is required only by certain element processors (e.g., see the sections on Processors ES1p and ES7p later in this chapter).

7.2.3.1.4 The GROUP Subcommand

The GROUP subcommand allows the user to break up the total set of elements of a particular type into groups. The subcommand format is:

GROUP = *group_number*

The *group_number* specified by this phrase remains in effect for all subsequent elements defined via the ELEMENT subcommand until another GROUP subcommand is issued. Group numbers should be assigned consecutively to elements. (Default: GROUP = 0)

7.2.3.1.5 The FAB_ID Subcommand

The FAB_ID subcommand is used to assign fabrication identification numbers to subsequently defined elements. The subcommand format is:

FAB_ID = *fabrication_number*

where the *fabrication_number* corresponds to the number assigned to the fabrication when it is defined via the Generic Constitutive Processor (GCP). (Default: FAB_ID = 1) The term *fabrication* refers to the combination of material and cross-sectional properties. For example, a layered shell fabrication contains information regarding the layer thicknesses, orientations, and stacking sequence, as well as the individual layer material numbers. Refer to Chapter 8, *Constitutive Processors*, for details.

7.2.3.1.6 The FAB_DIR Subcommand

The FAB_DIR subcommand may be used to orient the fabrication coordinate frame, x_f, y_f, z_f (see Section 2.2, *Reference Frames and Coordinate Systems*) relative to all subsequently defined elements. The command format is:

FAB_DIR = *fabrication_direction*

where the following options are provided.

fabrication_direction	Description
ELEMENT	The fabrication frame and local element stress frames coincide. (Default)
GLOBAL { X | Y | Z }	The fabrication x_f axis is parallel to the global x_g, y_g, or z_g axis if X, Y, or Z, respectively, is used in the subcommand. The fabrication z_f axis is parallel to the local element normal axis for shell elements, otherwise it is obtained by permuting the global axes. The fabrication y_f axis is defined by the right-hand rule.
POINT $\bar{x}$ [, θ]	A reference point, $\bar{x} = x_g, y_g, z_g$, is connected to each element integration point by a vector $\bar{v}$. The fabrication y_f axis is defined by taking the cross product of $\bar{v}$ and the element local normal vector. The fabrication z_f axis is parallel to the element local normal vector. The fabrication x_f axis is defined via the right-hand-rule. The optional angle θ is used to rotate x_f-y_f plane counter-clockwise about the z_f axis post-facto. (Relevant only for shells.)
VECTOR $\bar{v}$ [, θ]	The fabrication y_f axis is defined by taking the cross product of the user-specified vector, $\bar{v} = v^x_g, v^y_g, v^z_g$, and the element local normal vector. The fabrication z_f axis is assumed parallel to the element local normal vector. The fabrication x_f axis is defined via the right-hand-rule. The optional angle θ is used to rotate the x_f-y_f plane counter-clockwise about the z_f axis post-facto. (Relevant only for shells.)
PLANE $\bar{u}$, $\bar{v}$ [, θ]	The two user-specified vectors, $\bar{u}$ and $\bar{v}$, given in global components, are crossed to obtain the z_f axis. The x_f axis is parallel to $\bar{u}$. The y_f axis is obtained via the right-hand rule. For shell elements, the resulting triad is projected so that z_f aligns with the element normal, and the projected x_f-y_f axes are then rotated by an optional angle θ about z_f.
BEAM *node*	Node number of beam element reference point.

For an illustration of these options, refer to Section 2.7, *Orientation of Fabrication Reference Frames*.

7.2.3.1.7 The FAB_ECC Subcommand

The FAB_ECC subcommand may be used to offset the fabrication coordinate frame, x_f, y_f, z_f (see Section 2.2, *Reference Frames and Coordinate Systems*) relative to the element nodal reference surface The command format is:

FAB_ECC = *fabrication_eccentricity*

where the following options are provided. (Default: FAB_ECC = 0.)

Fabrication Type	*fabrication_eccentricity*	Description
Beam (1D)	e_y, e_z	The two eccentricities in the element local y- and z-axes of the beam cross-section relative to the spanwise axis connecting the element nodes. (Default: 0, 0)
Shell (2D)	e_z	The eccentricity in the element local normal direction of the shell thickness relative to the reference-surface connecting the element nodes. (Default: 0)
Solid (3D)	—	Irrelevant.

7.2.3.1.8 The SURFACE Subcommand

The SURFACE subcommand is relevant only if the qualifier /SOLID_MODEL=USER has been employed with the DEFINE ELEMENT command. It links individual elements with solid model surface IDs. The command format is:

SURFACE = *solid_model_surface_id*

where the *solid_model_surface_id* is a number defined by the user and referred to in user-written solid-model interface definition routines (see Chapter 16, *Solid Model Interface*). (Default: SURFACE = 1)

7.2.3.1.9 The ELEMENT (Nodal Connectivity) Subcommand

The ELEMENT subcommand defines nodal (and line) connectivity for each element of the type specified by the RESET ELEMENT_TYPE command. It must be used repeatedly until all such elements are defined. The subcommand format is:

ELEMENT = e NODES = $n_1, n_2, \ldots, n_{nen}$ [LINES = $l_1, l_2, \ldots, l_{nle}$]

where

Parameter	Description
e	Element number (does not have to be sequential). These element numbers are relative to a particular element processor/type combination. There are no "global" element numbers in COMET-AR.
n_i	Global node number of element node i.
nen	Number of element nodes.
l_i	Solid-model line ID associated with element boundary i. Relevant only if /SOLID_MODEL=USER qualifier is used with DEFINE ELEMENTS command.
nle	Number of lines (i.e., 1D boundaries) per element.

7.2.3.1.10 The END_DEFINE_ELEMENTS Subcommand

This subcommand terminates the element definition session for the current element type within the current processor. The subcommand format is:

END_DEFINE_ELEMENTS

7.2.3.1.11 Input Datasets

A summary of input datasets required by the DEFINE ELEMENTS command is given in Table 7.2-3.

Table 7.2-3 Input Datasets Required by DEFINE ELEMENTS Command

Dataset	Class	Contents
CSM.SUMMARY	CSM	Model summary dataset for the initial mesh (0). Only nodal summary parameters need be present.

7.2.3.1.12 Output Datasets

A summary of output datasets created or updated by the DEFINE ELEMENTS command is given in Table 7.2-4. Datasets marked with an asterisk are created if they don't exist; other datasets are simply modified.

Table 7.2-4 Output Datasets Created/Updated by DEFINE ELEMENTS Command

Dataset	Class	Contents
CSM.SUMMARY	CSM	Model summary dataset for the initial mesh; updated with element type attributes.
EltNam.DEFINITION*	EDT	Element definition dataset for all elements of current element type. Contains element nodal connectivity and related parameters. (*EltNam* denotes the concatenation of the current element processor name and the current element type name, i.e., *EltNam* = *EltProc_EltTyp*.)
EltNam.FABRICATION*	EFT	Element fabrication dataset for all elements of current element type. Contains element fabrication numbers, orientation (direction) options, and eccentricities.
EltNam.GEOMETRY*	EGT	Element solid-model geometry dataset for all elements of current element type. Relevant only if the qualifier /SOLID_MODEL=USER was used with the DEFINE ELEMENTS command.

7.2.3.2 The DEFINE LOADS Command

The DEFINE LOADS command can be used to define element loads for all elements of a particular type, and store them in the database for subsequent recovery during the solution phase. By element loads, we refer to distributed forces (e.g., line loads, pressures, and body forces) which require subsequent element processing to convert them into consistent nodal forces. The purpose of the DEFINE LOADS command is simply to store the primitive element load distributions in the database. Consistent element nodal forces can then be subsequently computed via the FORM FORCE/EXT command, discussed later in this section.

7.2.3.2.1 Command Syntax

The DEFINE LOADS command is composed of a nested group of subcommands and qualifiers that allows the definition of distributed loads of a particular type (e.g., line, surface, body), for all elements of a particular type. The meta-language description of the DEFINE LOADS command and its subcommands is as follows.

```
DEFINE LOADS  /Type    [ /LIVE ]   [ /SYSTEM = System ]
                [ GROUP = grp1, grp2, grpinc ]
                    [ ELEMENT = elt1, elt2, eltinc ]
                        [ Boundary = bnd1, bnd2, bndinc ]
                            [ NODE = nod1, nod2, nodinc ]
                                LOAD = load_values
                                :
END_DEFINE_LOADS
```

Individual qualifiers and subcommands are described in the following subsections. The above syntax involves "implied loops" on element group, element number, element boundary, and element node. The default range for these loops is everything: if the optional GROUP, ELEMENT, *Boundary*, and NODE subcommands are omitted, the specification of a load vector via the LOAD command would then be applied to all nodes of all boundaries of all elements of the current type. The current element type must be specified a priori via the RESET ELEMENT_TYPE command, described later in this section.

7.2.3.2.2 The /Type Qualifier

The /Type qualifier must be used to indicate the type of load that is to be defined. Valid options are described below.

Load *Type* Option	Description
/LINE	Line loads are defined as force (and/or moment) vectors per unit length. They may be applied to 1-D elements or along the edges of 2-D and 3-D elements.
/PRESSURE	Pressure loads are defined as forces per unit area that are directed normal to an element's surface. Positive pressure values are assumed to point along the "outward" normal. They are relevant only for 2D (plate/shell) elements and the 2-D surfaces of 3D elements.
/SURFACE	Surface loads are defined as general traction vectors (i.e., force or moment per unit element surface area). They may be applied to 2-D elements and to the 2-D surfaces of 3D elements.
/BODY	Body loads are defined as force vectors per unit mass, and may be applied to 1-D, 2-D and 3-D elements. A typical example of a body load is gravity, where the gravitational constant, g, is the magnitude, the direction is fixed (i.e., towards the earth), and both are constant for all nodes and elements in the structure.

7.2.3.2.3 The /SYSTEM Qualifier

The /SYSTEM qualifier indicates the name of the coordinate system in which the specified load components are to be interpreted. The qualifier format is:

```
/SYSTEM = System
```

where valid system names are

Load *System* Option	Description
GLOBAL (Default)	Indicates that the components of the load vector, specified via the LOAD subcommand, are expressed in the global-Cartesian coordinate system (x_g, y_g, z_g).
NODAL	Indicates that the components of the load vector are expressed in the computational frame (x_c, y_c, z_c) at each node. (See Section 2.2, *Reference Frames and Coordinate Systems*).
ELEMENT	Indicates that the components of the load vector are to be expressed in the element-Cartesian coordinate system (x_e, y_e, z_e). (This system is the same as the element corotational system and is fixed within an element.)

7.2.3.2.4 The /LIVE Qualifier

The optional /LIVE load qualifier may be used to designate element loads that are to be displacement dependent. Currently, the only type of live load implemented is the live pressure load, which is a pressure that remains normal to the element surface throughout deformation. A common example of this type of loading is the hydrostatic pressure applied to a submerged vehicle. To define live pressure loads, enter the command:

```
DEFINE LOADS /PRESSURE /LIVE
```

followed by the appropriate subcommands.

7.2.3.2.5 The GROUP Subcommand

The optional GROUP subcommand may be used to specify a range of element groups (i.e., a subset with the current element type) to be loaded by the subsequent LOAD subcommand. The command format is:

```
GROUP = grp1, grp2, grpinc
```

where the range parameters are defined as follows:

Parameter	Description
grp1	First element group in the range. (Default: 1)
grp2	Last element group in the range. (Default: Highest group number defined for current element type.)
grpinc	Increment used to count from *grp1* to *grp2*. (Default: 1)

7.2.3.2.6 The ELEMENT Subcommand

The optional ELEMENT subcommand may be used to specify a range of elements (i.e., a subset within the current element type) to be loaded by the subsequent LOAD subcommand. The command format is:

ELEMENT = *elt1, elt2, eltinc*

where the range parameters are defined as follows:

Parameter	Description
elt1	First element in the range. (Default: 1)
elt2	Last element in the range. (Default: Highest element number)
eltinc	Increment used to count from *elt1* to *elt2*. (Default: 1)

If the GROUP subcommand is set to all groups, then the element range parameters refer to the absolute element number within the group. Otherwise, the element range parameters refer to element numbers relative to the beginning of the specified groups.

7.2.3.2.7 The "Boundary" Subcommand

The optional *Boundary* subcommand may be used to specify a range of element boundaries to be loaded by the subsequent LOAD subcommand. The command format is:

Boundary = bnd1, bnd2, bndinc

where the range parameters are defined as follows:

Parameter	Description
Boundary	Boundary type name. Must be set to LINE for line loads or SURFACE for surface loads and pressures. Irrelevant for body loads.
bnd1	First element boundary in the range. (Default: 1)
bnd2	Last element boundary in the range. (Default: Highest element boundary number of type *Boundary*)
bndinc	Increment used to count from *bnd1* to *bnd2*. (Default: 1)

7.2.3.2.8 The NODE Subcommand

The optional NODE subcommand may be used to specify a range of element boundary nodes to be loaded by the subsequent LOAD subcommand. The subcommand format is:

NODE = *nod1, nod2, nodnc*

where the range parameters are defined as follows:

Parameter	Description
nod1	First element boundary node in the range. (Default: 1)
nod2	Last element boundary node in the range. (Default: Highest element boundary node for the boundaries specified via the *Boundary* subcommand)
nodinc	Increment used to count from *nod1* to *nod2*. (Default: 1)

The node numbers appearing in the NODE subcommand parameters are not global node numbers; they are element boundary node numbers (i.e., they are relative to each element boundary). For example, for a 4-node quadrilateral element, the element boundary node numbers on each of the four element boundaries would range from 1 to 2. (Refer to specific ES*i* section for numbering conventions.)

7.2.3.2.9 The LOAD Subcommand

The LOAD subcommand is used to specify the component(s) of the distributed load vector (or pressure), for the range of element groups, elements, element boundaries, and element boundary nodes indicated in the GROUP, ELEMENT, *Boundary*, and NODE subcommands, respectively. The subcommand format is:

LOAD = *load_values*

where the *load_values* are components defined according to the load */Type* qualifier as follows:

Load */Type*	*load_values*
/LINE	Vector of 3 nodal force components, and 3 nodal moment components (if applicable), per unit length, in the coordinates system (i.e., reference frame) indicated by the /SYSTEM qualifier.
/PRESSURE	Pressure magnitude (one number), with positive sign taken along the outward normal to the element surfaces specified (via the *Boundary* subcommand).
/SURFACE	Vector of 3 nodal force components, and 3 nodal moment components (if applicable), per unit area, in the coordinates system indicated by the /SYSTEM qualifier.
/BODY	Vector of 3 nodal force components (nodal body moments are not accepted), per unit mass, in the coordinate system indicated by the /SYSTEM qualifier. (These loads are scaled by the element mass density, defined via the Generic Constitutive Processor, when body loads are converted into consistent nodal forces during the FORM FORCE command.)

7.2.3.2.10 The END_DEFINE_LOADS Subcommand

This subcommand terminates the element load definition session for the current load type and element type within the current element processor. The subcommand format is:

```
END_DEFINE_LOADS
```

7.2.3.2.11 Input Datasets

A summary of input datasets required by the DEFINE LOADS command is given in Table 7.2-5.

Table 7.2-5 Input Datasets Required by DEFINE LOADS Command

Dataset	Class	Contents
CSM.SUMMARY	CSM	Model summary dataset for the initial mesh (0).
EltNam.DEFINITION	EDT	Element definition dataset for all elements of current element type, where *EltNam = EltProc_EltTyp* is defined via the RESET ELEMENT_TYPE command.

7.2.3.2.12 Output Datasets

A summary of output datasets created or updated by the DEFINE LOADS command is given in Table 7.2-6. Datasets marked with an asterisk are created if they don't exist; other datasets are simply modified.

Table 7.2-6 Output Datasets Created/Updated by DEFINE LOADS Command

Dataset	Class	Contents
EltNam.LOAD.*ldset**	ELT	Element load definitions for all elements of element type *EltTyp* within element processor *EltProc* (*EltNam=EltProc_EltTyp*) for load set *ldset*, where *EltNam* and *ldset* are defined via prior RESET commands. (See RESET command later in this section).

7.2.3.3 The DEFINE FREEDOMS Command

The DEFINE FREEDOMS command can be used to generate a table of potentially active nodal degrees of freedom (DOFs) based on all elements previously defined with an ES*i* processor. This table is output to the database as a Nodal Definition Table (NDT) dataset called ELEMENT_NODAL.DOFS. This dataset is first initialized then updated cumulatively by each ES*i* processor for which the DEFINE FREEDOMS command is invoked.

The DEFINE FREEDOMS command is automatically invoked via the INITIALIZE utility procedure (for all participating element processors), when the user selects the AUTO_DOF_SUP option at the solution procedure level.

7.2.3.3.1 Command Syntax

The format of the DEFINE FREEDOMS command is simply:

DEFINE FREEDOMS [= *elt_dof_dataset*]

where *elt_dof_dataset* is the name of the element nodal DOF table (class NDT), which defaults to NODAL.ELT_DOF.

7.2.3.3.2 Input Datasets

A summary of input datasets required by the DEFINE FREEDOMS command is given in Table 7.2-7.

Table 7.2-7 Input Datasets Required by DEFINE FREEDOMS Command

Dataset	Class	Contents
CSM.SUMMARY	CSM	Model summary dataset for the initial mesh (0).
EltNam.DEFINITION	EDT	Element definition dataset for all elements of current element type, where *EltNam* = *EltProc_EltTyp* is defined via the RESET ELEMENT_TYPE command.
NODAL.COORDINATE	NCT	Nodal coordinates.
NODAL.TRANSFORMATION	NTT	Nodal transformation matrices, representing orientation of computational coordinate system: x_c,y_c,z_c.

7.2.3.3.3 Output Datasets

A summary of output datasets created or updated by the DEFINE FREEDOMS command is given in Table 7.2-8. Datasets marked with an asterisk are created if they don't exist; other datasets are simply modified.

Table 7.2-8 Output Datasets Created/Updated by DEFINE FREEDOMS Command

Dataset	Class	Contents
NODAL.ELT_DOF*	NDT	Nodal DOF table indicating potentially active DOFs at nodes based on the type and orientation of the elements attached there. The attribute STATES indicates which nodal DOFs have sufficient element stiffness to be active. Such STATES are given the value qFREE; others are given the value qSPCz (for single-point constraint with value zero). This element_nodal DOF table may later be merged with the final nodal DOF table (which includes actual model boundary conditions) via processor COP, as is done automatically when the user selects the AUTO_DOF_SUP option from any solution procedure.

7.2.3.4 The DEFINE NORMALS Command

The DEFINE NORMALS command generates a dataset called NODAL.NORMALS which contains the average element-normal vectors for all shell elements attached to each node in the model. The average shell element normal vectors are used for defining drilling DOF stabilization flags (see the DEFINE DRILL_FLAGS command).

The DEFINE NORMALS command is automatically invoked by the INITIALIZE Utility Procedure (for all participating element processors) when the user selects either the AUTO_DRILL or AUTO_TRIAD option from a solution procedure (e.g. AR_CONTROL).

7.2.3.4.1 Command Syntax

The format of the DEFINE NORMALS command is simply:

```
DEFINE NORMALS
```

with no optional qualifiers or subcommands.

7.2.3.4.2 Input Datasets

A summary of input datasets that are required by the DEFINE NORMALS command is given in Table 7.2-9.

Table 7.2-9 Input Datasets Required by DEFINE NORMALS Command

Dataset	Class	Contents
CSM.SUMMARY	CSM	Model summary dataset for the initial mesh (0).
EltNam.DEFINITION	EDT	Element definition dataset for all elements of current element type, where *EltNam* = *EltProc_EltTyp* is defined via the RESET ELEMENT_TYPE command.
NODAL.COORDINATE	NCT	Nodal coordinates.
NODAL.TRANSFORMATION	NTT	Nodal transformation matrices, representing orientation of computational coordinate system: x_c,y_c,z_c.

7.2.3.4.3 Output Datasets

A summary of output datasets created or updated by the DEFINE NORMALS command is given in Table 7.2-10. Datasets marked with an asterisk are created if they don't exist; other datasets are simply modified.

Table 7.2-10 Output Datasets Created/Updated by DEFINE NORMALS Command

Dataset	Class	Contents
NODAL.NORMAL*	NAT	Table of average nodal normal (unit) vectors, ranging over all nodes in the model. Each column of the table thus consists of a vector, $\mathbf{n}_A = \{ n_A^x, n_A^y, n_A^z \}$, where A is the node number and x, y, z denote global-cartesian components of the normal vector. This table is initialized to zero and updated by accumulating contributions from all model shell-element processors. The accumulated normal vectors at each node are then normalized to become unit vectors by the last active element processor/type in the model.

7.2.3.5 The DEFINE DRILL_FLAGS Command

The DEFINE DRILL_FLAGS command generates or updates a dataset NODAL.DRILL_FLAGS, which contains an integer flag for each node in the model indicating whether or not stabilization is needed for the shell-element drilling rotation DOF at that node. This stabilization may take the form of artificial drilling stiffness (if the AUTO_DRILL solution procedure argument is <true>) or re-orientation of the computational directions (if the AUTO_TRIAD solution procedure argument is <true>).

The DEFINE DRILL_FLAGS command is invoked automatically via the INITIALIZE Utility Procedure (for every participating element processor) when the user selects either the AUTO_DRILL or AUTO_TRIAD option from a solution procedure (e.g., AR_CONTROL).

7.2.3.5.1 Command Syntax

The format of the DEFINE DRILL_FLAGS command is:

```
DEFINE DRILL_FLAGS
```

with no optional qualifiers or subcommands.

7.2.3.5.2 Input Datasets

A summary of input datasets required by the DEFINE DRILL_FLAGS command is given in Table 7.2-11.

Table 7.2-11 Input Datasets Required by DEFINE DRILL_FLAGS Command

Dataset	Class	Contents
CSM.SUMMARY	CSM	Model summary dataset for the initial mesh (0).
EltNam.DEFINITION	EDT	Element definition dataset for all elements of current element type, where *EltNam = EltProc_EltTyp* is defined via the RESET ELEMENT_TYPE command.
NODAL.COORDINATE	NCT	Nodal coordinates.
NODAL.NORMALS	NTT	Average nodal normals generated via the DEFINE NORMALS command.

7.2.3.5.3 Output Datasets

A summary of output datasets created or updated by the DEFINE DRILL_FLAGS command is given in Table 7.2-12. Datasets marked with an asterisk are created if they don't exist; others are simply modified.

Table 7.2-12 Output Datasets Created/Updated by DEFINE DRILL_FLAGS Command

Dataset	Type	Contents
NODAL.DRILL_FLAG*	NAT	Table of nodal drilling flags with integer values = 0 (false) if drilling stabilization is not required at the node, or 1 (true) if drilling stabilization is required.

7.2.3.6 The DEFINE ATTRIBUTES Command

The DEFINE ATTRIBUTES command may be used to define and store the database arbitrary element data, at element integration points, nodes, or centroids. Data is stored in an EAT (generic Element Attributes Table), called *EltNam.Attribute*, where the *Attribute* name is user-specified.

7.2.3.6.1 Command Syntax

The DEFINE ATTRIBUTES command syntax is shown below.

```
DEFINE ATTRIBUTES  [ ldi, ]  Attrib_name  /NUMBER=num_attrib  /LOC = attrib_loc
                [ GROUPS = grp1, grp2, grpinc ]
                        [ ELEMENTS = elt1, elt2, eltinc ]
                                [ POINTS = pt1, pt2, ptinc ]
                                        ATTRIBUTES = att1, att2, . . ., num_attrib
                                                :
END_DEFINE_ATTRIBUTES
```

where the individual qualifiers and subcommands are described below.

Parameter	Description
ldi	Logical device index of computational database. (Default: 1)
Attrib_name	Attribute name; dataset name = *EltNam.Attrib_name*
num_attrib	Number of attributes (i.e., variables) per element point
attrib_loc	Location name: INTEG_PTS, NODES, or CENTROIDS. (Default: CENTROIDS)
grp1, grp2, grpinc	First, last, and increment in element group range. (Default: All groups)
elt1, elt2, eltinc	First, last, and increment in element range. If all groups are specified, then *elt1* and *elt2* refer to the absolute element number within the current element type. Otherwise, *elt1* and *elt2* refer to the relative element number within each specified group. (Default: All elements in groups specified by GROUPS subcommand)
pt1, pt2, ptinc	First, last, and increment in element point range. Element points refer to element nodes if *attrib_loc*=NODES, or element integration points if *attrib_loc*=INTEG_PTS. Irrelevant for attrib_loc=CENTROIDS. (Default: All points of type *attrib_loc* with specified element range)
att1, att2, . . ., num_attrib	List of attributes 1 through *num_attrib* to be stored at the elements/locations indicated by the GROUP, ELEMENT, and POINT subcommands.

The command syntax involves implied loops on element groups, element numbers, and element points. The default range for these loops is everything: if the optional GROUP, ELEMENT, or POINT subcommands are omitted, the specification of element attributes via the ATTRIBUTE subcommand would then be applied to all points of all elements of the current type. The current element type must be specified a priori via the RESET ELEMENT_TYPE command.

7.2.3.6.2 The END_DEFINE_ATTRIBUTES Subcommand

This subcommand terminates the element attribute definition session for the current attribute type and element type within the current element processor. The subcommand format is:

END_DEFINE_ATTRIBUTES

7.2.3.6.3 Input Datasets

A summary of input datasets required by the DEFINE ATTRIBUTES command is given in Table 7.2-13.

Table 7.2-13 Input Datasets Required by the DEFINE ATTRIBUTES Command

Dataset	Class	Contents
CSM.SUMMARY	CSM	Model summary dataset for the initial mesh (0).
EltNam.DEFINITION	EDT	Element definition dataset for all elements of current type, where *EltNam* = *EltProc_EltTyp* is defined via the RESET ELEMENT_TYPE command.

7.2.3.6.4 Output Datasets

A summary of output datasets created or updated by the DEFINE ATTRIBUTES command is given in Table 7.2-14. Datasets marked with an asterisk are created if they don't exist; others are simply modified.

Table 7.2-14 Output Datasets Created/Updated by DEFINE LOADS Command

Dataset	Class	Contents
*EltNam.Attrib_Name**	EAT	Table of element attributes. The location (i.e., points) within the element and the number of attributes at each point are specified via the DEFINE ATTRIBUTES command parameters *attrib_loc* and *num_attrib*, as is the attribute name, *Attrib_Name*. (All attributes created by this command are stored as floating point values.)

7.2.4 ES Processor INITIALIZE Command

The INITIALIZE command must be used between the model definition (DEFINE commands) and solution (FORM commands) phases of the analysis to generate or initialize certain datasets that are employed in subsequent solution tasks. The INITIALIZE command accomplishes three functions:

1) Creation of element interpolation datasets (i.e., *EltNam*.INTERPOLATION), required during later element error estimation;

2) Creation of element auxiliary storage datasets (i.e., *EltNam*.AUX_STORAGE), optionally required by certain element (ES*i*) processors; and

3) Initialization of constitutive datasets via the embedded generic constitutive processor (GCP).

The INITIALIZE command is automatically invoked by most Solution Procedures (via the INITIALIZE and ES Utility Procedures) at the beginning of the analysis and after every adaptive mesh update.

7.2.4.1 Command Syntax

The format of the INITIALIZE command is:

```
INITIALIZE
```

with no optional command qualifiers or subcommands.

7.2.4.1.1 Input Datasets

A summary of input datasets required by the INITIALIZE command is given in Table 7.2-15.

Table 7.2-15 Input Datasets Required by the INITIALIZE Command

Dataset	Class	Contents
CSM.SUMMARY	CSM	Model summary dataset for the initial mesh (0).
EltNam.DEFINITION	EDT	Element definition dataset for all elements of current element type, where *EltNam* = *EltProc_EltTyp* is defined via the RESET ELEMENT_TYPE command.
NODAL.COORDINATE	NCT	Nodal coordinate table.
NODAL.TRANSFORMATION	NTT	Nodal transformation (global-to-computational) table.
Constitutive Datasets	GCP	Material and fabrication properties and pointers.

7.2.4.1.2 Output Datasets

A summary of output datasets created or updated by the INITIALIZE command is given in Table 7.2-16. Datasets marked with an asterisk are created.

Table 7.2-16 Output Datasets Created/Updated by INITIALIZE Command

Dataset	Class	Contents
EltNam.INTERPOLATION*	EIT	Element interpolation, extrapolation, and numerical integration data; necessary for subsequent error estimation and post-processing.
EltNam.AUX_STORAGE*	EAT	Element auxiliary data; required only by certain element (ES*i*) processors.

Table 7.2-16 Output Datasets Created/Updated by INITIALIZE Command (Continued)

Dataset	Class	Contents
*Constitutive Datasets**	GCP	Various constitutive datasets and files managed by the GCP are opened and initialized by this command. For example, the initial integrated constitutive matrix is computed and stored for beam/shell elements with linear material properties. For nonlinear materials, historical data files are opened and initialized. (See Chapter 8, *Constitutive Processors*)

7.2.5 ES Processor FORM Commands

FORM commands (Table 7.2-17) are used to form the element arrays required during the solution phase of a COMET-AR analysis. These arrays include element force vectors, stiffness, and mass matrices which are employed during the primary solution phase to obtain a global displacement solution; and element strains, stresses, and strain energy densities, which can be computed after the displacement solution has been obtained (i.e., during the secondary solution phase).

Table 7.2-17 Summary of ES FORM Commands

FORM Command	Function
FORM FORCE [/INT/EXT/RES]	Forms and assembles element force vectors.
FORM STIFFNESS [/MATL/GEOM/LOAD/TANG]	Forms element stiffness matrices.
FORM MASS [/CONS/DIAG]	Forms element mass matrices; assembles if diagonal matrix.
FORM STRAIN	Forms element strains.
FORM STRESS	Forms element stresses.
FORM STRAIN_ENERGY	Forms elt. strain energy densities.

All of the FORM commands may be invoked indirectly via the ES Utility Procedure, which automatically runs all relevant ES*i* processors and element types with a single procedure call.

7.2.5.1 The FORM FORCE Command

The FORM FORCE command is used to form element force vectors (internal, external, or residual) for all elements of a given type (as specified a priori by the RESET ELEMENT_TYPE command) within the currently running element (ES*i*) processor. Element force vectors are not stored in the database, but rather assembled directly into a system force vector.

7.2.5.1.1 Command Syntax

The FORM FORCE command has the following syntax:

```
FORM FORCE [ /Qualifier ]
```

where the following are valid command qualifiers.

Qualifier	Description
INTERNAL	Indicates element internal force vectors are to be formed and assembled into a system force vector. The internal force vector is defined as the set of element nodal forces which depends explicitly on the element internal stress distribution (and possibly on initial strains or temperatures). In a conservative system, this vector emanates from the first variation of the element strain energy functional. (For statistics problems, the internal force vector is equivalent (in both magnitude and direction) to the external force vector at nodes where external forces are applied, and equivalent to reaction forces at nodes where displacements are applied.)
EXTERNAL	Indicates element external force vectors are to be formed and assembled into a system force vector. The external force vector is defined as the set of consistent element nodal forces corresponding to the distributed loads specified via the DEFINE LOADS command.
RESIDUAL (Default)	Indicates element residual force vectors are to be formed and assembled into a system force vector. The residual force vector is defined as the difference between the external and internal force vectors, i.e., $\mathbf{f}^{res} = \mathbf{f}^{ext} - \mathbf{f}^{int}$

7.2.5.1.2 Input Datasets

Input datasets required by the FORM FORCE commands are listed in Table 7.2-18.

Table 7.2-18 Input Datasets Required by the FORM FORCE Commands

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset for current mesh (see RESET MESH command).
EltNam.DEFINITION...*mesh*	EDT	Element definition dataset for all elements of current element type. (See RESET ELEMENT_TYPE command).
EltNam.FABRICATION...*mesh*	EFT	Element fabrication table; includes element fabrication numbers, orientation, and eccentricity.
EltNam.LOAD.*ldset*..*mesh*	ELT	Element load table for load set number *ldset*, as specified via the RESET LOAD_SET command. (This dataset is irrelevant for INTERNAL forces.)
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate table.
NODAL.TRANSFORMATION...*mesh*	NTT	Nodal transformation table (global-to-computational transformation matrices or triads).

Table 7.2-18 Input Datasets Required by the FORM FORCE Commands (Continued)

Dataset	Class	Contents
Nodal Displacement Dataset (Default name: NODAL.DISPLACEMENT.*step..mesh*)	NVT	Nodal displacement vector table. Dataset name may be reset via RESET DISPLACEMENT command.
Nodal Rotation Dataset (Default name: NODAL.ROTATION.*step..mesh*)	NAT	Nodal rotation (pseudovector) table. Dataset name may be reset via RESET ROTATION command.
Constitutive Datasets	GCP	Material and fabrication datasets, and constitutive historical data (if necessary) managed by the GCP (see Chapter 8, *Constitutive Processors*).

7.2.5.1.3 Output Datasets

Output datasets created/updated by the FORM FORCE command are listed in Table 7.2-19. Datasets marked with an asterisk are created if they don't exist.

Table 7.2-19 Output Datasets Created/Updated by FORM FORCE Command

Dataset	Class	Contents
*Nodal Force Dataset** (Default name: NODAL.FORCE)	NVT	Assembled nodal force vectors, containing INTERNAL, EXTERNAL, or RESIDUAL forces, depending on command qualifier.

7.2.5.2 The FORM STIFFNESS Command

The FORM STIFFNESS command is used to form element stiffness matrices (material, geometric, load, or tangent) for all elements of the type pre-specified by the RESET ELEMENT_TYPE command, within the currently running ES*i* processor. Element stiffness matrices are stored in the database in an element matrix table (EMT) dataset, for subsequent system assembly.

7.2.5.2.1 Command Syntax

The FORM STIFFNESS command has the following syntax:

```
FORM STIFFNESS [ /Qualifier ]
```

where the following are valid command qualifiers:

Qualifier	Description
MATERIAL	Indicates element material stiffness matrices are to be formed and stored in the database. The material stiffness matrix, $\mathbf{K}^{matl}$, is defined as that part of the tangent (or total) stiffness matrix which depends explicitly on material properties via the linearized material constitutive (stress/strain) matrix. For linear response analysis, $\mathbf{K}^{matl}$ is equivalent to the tangent stiffness, $\mathbf{K}^{tang}$.
GEOMETRIC	Indicates element geometric stiffness matrices are to be formed and stored in the database. The geometric stiffness matrix, $\mathbf{K}^{geom}$, is defined as that part of the tangent (or total) stiffness matrix which depends explicitly on stresses. It is obtained by linearization of the strain-displacement relations, and is often called the initial-stress stiffness matrix. It is needed for both buckling eigenvalue analysis and for nonlinear analysis.
LOAD	Indicates element load stiffness matrices are to be formed. The load stiffness matrix, $\mathbf{K}^{load}$, is defined as that part of the tangent stiffness matrix emanating from displacement-dependent loads (e.g., live pressures and other follower forces). It is needed only for certain linear buckling problems in which follower forces affect the critical load. For nonlinear analysis it is typically only of marginal importance. (The LOAD stiffness option is currently untested in COMET-AR.)
TANGENT (Default)	Indicates element tangent stiffness matrices are to be formed and stored in the database. The tangent (or total) stiffness matrix, $\mathbf{K}^{tang}$, is defined as the derivative of the element residual force vector with respect to the element displacement vector, i.e., $$\mathbf{K}^{tang} = \frac{\partial}{\partial \mathbf{d}}\mathbf{f}^{int} - \frac{\partial}{\partial \mathbf{d}}\mathbf{f}^{ext} = (\mathbf{K}^{matl} + \mathbf{K}^{geom}) + \mathbf{K}^{load}$$ which includes all pertinent effects (material, geometric, and load stiffnesses) and hence should be used in conjunction with any form of nonlinear analysis, including eigenanalysis about a nonlinear load state.

7.2.5.2.2 Input Datasets

Input datasets required by the FORM STIFFNESS command are listed in Table 7.2-20.

Table 7.2-20 Input Datasets Required by the FORM STIFFNESS Commands

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset for current mesh (see RESET MESH command).
EltNam.DEFINITION...*mesh*	EDT	Element definition dataset for all elements of current element type. (See RESET ELEMENT_TYPE command.)
EltNam.FABRICATION...*mesh*	EFT	Element fabrication table; includes element fabrication numbers, orientation, and eccentricity.
EltNam.LOAD.*ldset..mesh*	ELT	Element load table for load set number *ldset*, as specified via the RESET LOAD_SET command. (This dataset is used only if the load stiffness is included via the /LOAD qualifier or the RESET NL_LOAD command.)

Table 7.2-20 Input Datasets Required by the FORM STIFFNESS Commands (Continued)

Dataset	Class	Contents
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate table.
NODAL.TRANSFORMATION...*mesh*	NTT	Nodal transformation table (global-to-computational transformation matrices or triads).
Nodal Displacement Dataset (Default name: NODAL.DISPLACEMENT.*step..mesh*)	NVT	Nodal displacement vector table. Dataset name may be reset via RESET DISPLACEMENT command. Relevant only for geometrically nonlinear analysis (see RESET NL_GEOM command) or in conjunction with the /GEOMETRIC stiffness qualifier.
Nodal Rotation Dataset (Default name: NODAL.ROTATION.*step.mesh*)	NAT	Nodal rotation (pseudovector) table. Dataset name may be reset via RESET ROTATION command. Relevant only for geometrically nonlinear analysis (see RESET NL_GEOM command).
Constitutive Datasets	GCP	Material and fabrication datasets, and constitutive historical data (if necessary), managed by the GCP (see Chapter 8, *Constitutive Processors*).

7.2.5.2.3 Output Datasets

Output datasets created/updated by the FORM STIFFNESS command are listed in Table 7.2-21. Datasets marked with an asterisk are created if they don't exist.

Table 7.2-21 Output Datasets Created/Updated by FORM STIFFNESS Command

Dataset	Class	Contents
*Element Matrix Dataset** (Default name: *EltNam*.STIFFNESS...*mesh*)	EMT	Element stiffness matrices of type material, geometric, load, or tangent, depending on the command qualifier. These matrices are stored in symmetric, upper triangular form. The dataset name may be changed via the RESET STIFFNESS command.

7.2.5.3 The FORM MASS Command

The FORM MASS command is used to form element mass matrices (consistent or lumped) for all elements of the type pre-specified by the RESET ELEMENT_TYPE command, within the currently running ES*i* processor. Element consistent mass matrices are stored in the database in an element matrix table (EMT) dataset for subsequent system assembly. Element lumped (i.e., diagonal) mass matrices are assembled directly into a nodal vector (NVT) dataset representing the entire model.

7.2.5.3.1 Command Syntax

The FORM MASS command has the following syntax:

```
FORM MASS [ /Qualifier ]
```

where the following are valid command qualifiers:

Qualifier	Description
CONSISTENT (Default)	Indicates element consistent mass matrices are to be formed and output to the database in an element matrix table (EMT) dataset.
DIAGONAL	Indicates element diagonal (lumped) mass matrices are to be formed and assembled directly into a nodal vector table (NVT) dataset

7.2.5.3.2 Input Datasets

Input datasets required by the FORM MASS command are listed in Table 7.2-22.

Table 7.2-22 Input Datasets Required by the FORM MASS Commands

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset for current mesh (see RESET MESH command).
EltNam.DEFINITION...*mesh*	EDT	Element definition dataset for all elements of current element type. (See RESET ELEMENT_TYPE cmd.)
EltNam.FABRICATION...*mesh*	EFT	Element fabrication table; includes element fabrication numbers, orientation and eccentricity.
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate table.
NODAL.TRANSFORMATION...*mesh*	NTT	Nodal transformation table (global-to-computational transformation matrices or triads).
Constitutive Datasets	GCP	Mass properties, such as density and mass moments of inertia, depending on element/fabrication type. (See Chapter 8, *Constitutive Processors*)

7.2.5.3.3 Output Datasets

Output datasets created/updated by the FORM MASS command are listed in Table 7.2-23. Datasets marked with an asterisk are created if they don't already exist.

Table 7.2-23 Output Datasets Created/Updated by FORM MASS Command

Dataset	Class	Contents
*Element Matrix Dataset** (if /CONSISTENT mass)	EMT	Element consistent mass matrices, stored in symmetric, upper triangular form. (Default dataset name: *EltNam*.MASS...*mesh*)
*Nodal Vector Dataset** (if /DIAGONAL mass)	NVT	Assembled diagonal mass matrices in a nodal vector format. (Default dataset name: NODAL.DIAG_MASS...*mesh*)

7.2.5.4 The FORM STRAIN Command

The FORM STRAIN command is used to compute element strains for all elements of a given type (as specified a priori by the RESET ELEMENT_TYPE command) within the currently running element (ES*i*) processor. Element strains are stored in the database in an element stress/strain table (EST) dataset. They may be computed and stored at element integration points, element nodes, or element centroids, depending on the RESET STR_LOCATION command. The reference frame (x_s,y_s,z_s) in which the strain components are expressed depends on the RESET STR_DIRECTION command; whether they are pointwise or resultant quantities depends on the element type.

7.2.5.4.1 Command Syntax

The FORM STRAIN command has the following syntax:

```
FORM STRAIN
```

with no command qualifiers or subcommands.

7.2.5.4.2 Input Datasets

Input datasets required by the FORM STRAIN command are listed in Table 7.2-24.

Table 7.2-24 Input Datasets Required by the FORM STRAIN Commands

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset for current mesh (see RESET MESH command).
EltNam.DEFINITION...*mesh*	EDT	Element definition dataset for all elements of current element type. (See RESET ELEMENT_TYPE command)
EltNam.FABRICATION...*mesh*	EFT	Element fabrication table; includes element fabrication numbers, orientation and eccentricity.
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate table.

Table 7.2-24 Input Datasets Required by the FORM STRAIN Commands (Continued)

Dataset	Class	Contents
NODAL.TRANSFORMATION...*mesh*	NTT	Nodal transformation table (global-to-computational transformation matrices, or triads).
Nodal Displacement Dataset (Default name: NODAL.DISPLACEMENT.*step..mesh*)	NVT	Nodal displacement vector table. Dataset name may be reset via the RESET DISPLACEMENT command. (See also the RESET STEP and RESET MESH commands for definition of *step* and *mesh* numbers in the dataset name.)
Nodal Rotation Dataset (Default name: NODAL.DISPLACEMENT.*step..mesh*)	NAT	Nodal rotation (pseudovector) table. Dataset name may be reset via the RESET ROTATION command. (See also the RESET STEP and RESET MESH commands.)

7.2.5.4.3 Output Datasets

Output datasets created/updated by the FORMSTRAIN command are listed in Table 7.2-25. Datasets marked with an asterisk are created if they don't already exist; others are simply updated.

Table 7.2-25 Output Datasets Created/Updated by FORM STRAIN Command

Dataset	Class	Contents
Element Strain Dataset (Default name: *EltNam*.STRAIN.*step..mesh*)	EST	Element strains expressed in the coordinate system indicated by the RESET STR_DIRECTION command. Element strains may be computed and stored either at element integration points (STR attribute), element nodes (STRNOD attribute), or element centroids (STRCEN attribute), depending on the RESET STR_LOCATION command. All three locations (attributes) may be stored in the same dataset, via three separate applications of the FORM STRAIN command. The dataset name may be changed by reseting STRAIN, LOAD_SET, CONSTRAINT_SET, STEP, and/or MESH via the RESET command.

7.2.5.5 The FORM STRESS Command

The FORM STRESS command is used to compute element stresses for all elements of a given type (as specified a priori by the RESET ELEMENT_TYPE command) within the currently running element (ES*i*) processor. Element stresses are stored in the database in an element stress/strain table (EST) dataset. They may be computed and stored at element integration points, element nodes, or element centroids, depending on the RESET STR_LOCATION command. The reference frame (x_s,y_s,z_s) in which the stress components are expressed depends on the RESET STR_DIRECTION command; whether they are pointwise or resultant quantities depends on the element type.

7.2.5.5.1 Command Syntax

The FORM STRESS command has the following syntax:

```
FORM STRESS
```

with no command qualifiers or subcommands.

7.2.5.5.2 Input Datasets

Input datasets required by the FORM STRESS command are listed in Table 7.2-26.

Table 7.2-26 Input Datasets Required by the FORM STRESS Commands

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset for current mesh (see RESET MESH command).
EltNam.DEFINITION...*mesh*	EDT	Element definition dataset for all elements of current element type. (See RESET ELEMENT_TYPE cmd.)
EltNam.FABRICATION...*mesh*	EFT	Element fabrication table; includes element fabrication numbers, orientation and eccentricity.
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate table.
NODAL.TRANSFORMATION...*mesh*	NTT	Nodal transformation table (global-to-computational transformation matrices, or triads).
Nodal Displacement Dataset (Default name: NODAL.DISPLACEMENT.*step..mesh*)	NVT	Nodal displacement vector table. Dataset name may be reset via the RESET DISPLACEMENT command. (See also the RESET STEP and RESET MESH commands.)
Nodal Rotation Dataset (Default name: NODAL.DISPLACEMENT.*step..mesh*)	NAT	Nodal rotation (pseudovector) table. Dataset name may be reset via the RESET ROTATION command. (See also the RESET STEP and RESET MESH commands.)

7.2.5.5.3 Output Datasets

Output datasets created/updated by the FORM STRESS command are listed in Table 7.2-27. Datasets marked with an asterisk are created if they don't already exist; others are simply updated.

Table 7.2-27 Output Datasets Created/Updated by FORM STRESS Command

Dataset	Class	Contents
*Element Stress Dataset** (Default name: *EltNam*.STRESS.*step..mesh*)	EST	Element stresses expressed in the coordinate system indicated by the RESET STR_DIRECTION command. Element stresses may be computed and stored either at element integration points (STR attribute), element nodes (STRNOD attribute), or element centroids (STRCEN attribute), depending on the RESET STR_LOCATION command. All three locations (attributes) may be stored in the same dataset, via three separate applications of the FORM STRAIN command. The dataset name may be changed by reseting STRESS, STEP, and/or MESH via the RESET command.

7.2.5.6 The FORM STRAIN_ENERGY Command

The FORM STRAIN_ENERGY command is used to compute element strain energy densities for all elements of a given type (as specified a priori by the RESET ELEMENT_TYPE command) within the currently running element (ES*i*) processor. Element strain energy densities are stored in the database in an element stress/strain table (EST) dataset. They may be computed and stored at element integration points, element nodes or element centroids, depending on the RESET STR_LOCATION command; whether they are pointwise or resultant (i.e., pre-integrated over the cross-section) quantities depends on the element type. For example, pointwise strain energy densities will be in units of strain energy per unit volume for continuum (3D) elements; and resultant strain energy densities would be in units of strain energy per unit reference-surface area for shell (2D) elements, and per unit reference-axis length for beam (1D) elements.

7.2.5.6.1 Command Syntax

The FORM STRAIN_ENERGY command has the following syntax:

```
FORM STRAIN_ENERGY
```

with no command qualifiers or subcommands.

7.2.5.6.2 Input Datasets

Input datasets required by the FORM STRAIN_ENERGY command are listed in Table 7.2-28.

Table 7.2-28 Input Datasets Required by the FORM STRAIN_ENERGY Commands

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset for current mesh (see RESET MESH command).
EltNam.DEFINITION...*mesh*	EDT	Element definition dataset for all elements of current element type. (See RESET ELEMENT_TYPE cmd.)
EltNam.FABRICATION...*mesh*	EFT	Element fabrication table; includes element fabrication numbers, orientation, and eccentricity.
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate table.
NODAL.TRANSFORMATION...*mesh*	NTT	Nodal transformation table (global-to-computational transformation matrices or triads).
Nodal Displacement Dataset (Default name: NODAL.DISPLACEMENT.*step..mesh*)	NVT	Nodal displacement vector table. Dataset name may be reset via the RESET DISPLACEMENT command. (See the RESET STEP and RESET MESH commands.)
Nodal Rotation Dataset (Default name: NODAL.DISPLACEMENT.*step..mesh*)	NAT	Nodal rotation (pseudovector) table. Dataset name may be reset via RESET ROTATION command. (See the RESET STEP and RESET MESH commands.
Constitutive Datasets	GCP	Material and fabrication datasets; and constitutive historical data files if nonlinear materials are present (see Chapter 8).

7.2.5.6.3 Output Datasets

Output datasets created/updated by the FORM STRAIN_ENERGY command are in Table 7.2-29.

Table 7.2-29 Output Datasets Created/Updated by FORM STRAIN_ENERGY Command

Dataset	Class	Contents
Element Strain Dataset (Default name: *EltNam*.STRAIN_ENERGY.*step..mesh*)	EST	Element strain energy densities, defined as $\hat{U} = \int_{\varepsilon} \sigma \bullet d\varepsilon$ where σ and ε denote stress and strain (resultants). For linear-elastic materials: $\hat{U} = \frac{1}{2}\sigma \bullet \varepsilon$ Element strain energy densities may be computed and stored at element integration points (STR attribute), element nodes (STRNOD attribute), or element centroids (STRCEN attribute), depending on the RESET STR_LOCATION command. All three locations may be stored in the same dataset via three separate applications of the FORM STRAIN_ENERGY command. The dataset name may be changed by reseting STRAIN_ENERGY, STEP, and/or MESH via the RESET command.

7.2.6 ES Processor RESET Commands

RESET commands are used to change dataset names and/or selected input parameters from their default values. Some RESET commands, such as RESET ELEMENT_TYPE, have no default settings and must always be used before employing other ES commands, such as DEFINE and FORM. A summary of RESET commands currently available to element (ES*i*) processors constructed via the generic element processor (ES) is given in Table 7.2-30.

Table 7.2-30 Summary of ES "RESET" Commands

RESET Command	Function
RESET COROTATION	Changes default element corotational option
RESET DISPLACEMENT	Changes default name of nodal displacement dataset
RESET DRILL_STIFF	Changes default value of artificial drilling stiffness parameter
RESET DRILL_TOL	Changes default value of drilling stabilization angle tolerance
RESET ELEMENT_TYPE	Sets element type name (*EltTyp*); required for all commands **(Mandatory prerequisite to all other ES commands.)**
RESET FORCE	Changes default name of nodal force dataset
RESET GCP_LDI	Changes default *ldi* of GCP material and fabrication datasets
RESET LDI	Changes default *ldi* of computational database library
RESET LOAD_FACTOR	Changes default load factor to be applied to element loads
RESET LOAD_SET	Changes default load set number for element loads
RESET MASS	Changes default name of output mass matrix dataset
RESET MESH	Sets/resets mesh number
RESET NL_GEOM	Changes default geometric nonlinearity option
RESET NL_LOAD	Changes default load nonlinearity option
RESET NL_MATL	Changes default material nonlinearity option
RESET PARAMETERS	Sets values of optional element research parameters
RESET PROJECTION	Changes default element projection option
RESET ROTATION	Changes default name of nodal rotation pseudovector dataset
RESET STEP	Sets/resets load- or time-step number
RESET STIFFNESS	Changes default name of element stiffness dataset
RESET STRAIN	Changes default name of element strain dataset
RESET STRAIN_ENERGY	Changes default name of element strain energy dataset
RESET STRESS	Changes default name of element stress dataset
RESET STR_DIR	Changes default stress/strain output coordinate system
RESET STR_LOC	Changes default stress/strain output locations

7.2.6.1 The RESET COROTATION Command

This command is used to change the default element corotational option for geometrically nonlinear analysis. The corotational capability is built in to the generic element processor (ES) and enables beam and shell elements to be employed with arbitrarily large rotations (but small to moderate strains) even if the element strain-displacement relations do not intrinsically account for large rotations (see the chapter on Corotation in Reference [1] for details). The command format is:

RESET COROTATION = *corotation_option*

where

corotation_option	Description
0	Element corotation will not be used. (Default)
1	Basic element corotation will be used. This option is sufficient unless True-Newton iteration is begin performed at the nonlinear solution procedure level.
2	Higher-order element corotation will be used. This option should be used only if True-Newton iteration has been selected at the nonlinear solution procedure level and may provide only marginal improvement in nonlinear convergence over option 1. It adds additional terms to the tangent stiffness matrix that render it more consistent.

The RESET COROTATION command is relevant before the following action commands: FORM STIFFNESS, FORM FORCE, FORM STRESS, FORM STRAIN, and FORM STRAIN_ENERGY; and only if geometrically nonlinear analysis (see RESET NL_GEOM command).

7.2.6.2 The RESET DISPLACEMENT Command

This command is used to change the default *ldi* and name of the nodal displacement dataset command. The command format is:

RESET DISPLACEMENT = [*ldi*,] *ds_name*

where *ldi* is the logical device index of the computational database and *ds_name* is the new dataset name. The default setting is:

RESET DISPLACEMENT = *ldi*, NODAL.DISPLACEMENT.*step.mesh*

where *ldi* is the *ldi* defined via the RESET LDI command, *step* is the load/time-step number defined via the RESET STEP command, and *mesh* is the mesh number defined via the RESET MESH command. For example, the command:

RESET DISPLACEMENT = NODAL.INC_DISP.*step*

is typically used in nonlinear analysis to label incremental displacement vectors at a step number *step*.

7.2.6.3 The RESET DRILL_STIFF Command

This command is used to change the default artificial drilling rotational stiffness option for (certain) shell element types. The command format is:

```
RESET DRILL_STIFF = Option [ , scale ]
```

where *Option* is either 1 (true) or 0 (false), and *scale* is an integer scale factor that depends on the particular element type. The default setting is:

```
RESET DRILL_STIFF = 0, 0
```

which implies that drilling stiffness will not be applied. (Relevant only before the FORM STIFFNESS command.)

7.2.6.4 The RESET DRILL_TOL Command

This command is used to change the default artificial drilling tolerance option for (certain) shell element types. The command format is:

```
RESET DRILL_TOL = angle
```

where *angle* is an integer angle tolerance indicating when some form of stabilization is required for shell element drilling rotational freedoms. If the angle between the shell-element normal and the average element normal (or a computational axis) at a node is less than this value, drilling stabilization may be required (depending on the element type). The default setting is:

```
RESET DRILL_TOL = 0
```

which implies that the internal default values provided by specific shell-element processors will be employed. (Relevant only for the DEFINE FREEDOMS and DEFINE DRILL_FLAGS commands.)

7.2.6.5 The RESET ELEMENT_TYPE Command (*Mandatory*)

This mandatory command is used to indicate which element type within a given ES*i* processor to operate on during subsequent DEFINE or FORM commands. The command format is:

```
RESET ELEMENT_TYPE = EltTyp
```

where *EltTyp* is the element type name. The full element name, *EltNam*, is automatically constructed by concatenating the current element processor name with the element type name, i.e.,

EltNam = EltProc_EltTyp

where *EltProc* is the processor name. *EltNam* is employed in the construction of many ES dataset names. **(This command is prerequisite for all other ES commands.)**

7.2.6.6 The RESET FORCE Command

This command is used to change the default *ldi* and name of the nodal force dataset. The command format is:

RESET FORCE = [*ldi*,] *ds_name*

where *ldi* is the logical device index of the computational database and *ds_name* is the new dataset name. The default setting is:

RESET FORCE = ldi, NODAL.FORCE.*step.mesh*

where *ldi* is the logical device index defined via the RESET LDI command, *ldset* is the load-set number defined via the RESET LOAD_SET command, and *mesh* is the mesh number defined via the RESET MESH command. For example, the command:

RESET FORCE = NODAL.INT_FORCE.*step*

is typically used in nonlinear analysis to label internal force vectors by load-step number, where *step* denotes the current load or time step number.

7.2.6.7 The RESET GCP_LDI Command

This command is used to change the default database logical device index (*ldi*) associated with all datasets managed by the Generic Constitutive Processor. The command format is:

RESET GCP_LDI = *gcp_ldi*

where *gcp_ldi* is the logical device index. The default setting is:

RESET GCP_LDI = 1

7.2.6.8 The RESET LOAD_FACTOR Command

This command is used to change the default load factor to be applied to all element loads. The command format is:

```
RESET LOAD_FACTOR = load_factor
```

where *load_factor* is a floating-point scale factor. The default setting is:

```
RESET LOAD_FACTOR = 1.0
```

Relevant before the following ES commands: FORM FORCE/EXT, FORM FORCE/RES, or FORM STIFFNESS/LOAD.

7.2.6.9 The RESET LOAD_SET Command

This command is used to change the default load set number for element loads during either load definition or consistent external force formation. The command format is:

```
RESET LOAD_SET = load_set
```

where *load_set* is an integer load-set number. The default setting is:

```
RESET LOAD_SET = 1
```

Relevant before the following ES commands: DEFINE LOADS, FORM FORCE/EXT, FORM FORCE/RES, or FORM STIFFNESS/LOAD.

7.2.6.10 The RESET LDI Command

This command is used to change the default logical device index (*ldi*) for all datasets input/output by the current ES*i* processor, except those for which an explicit *ldi* is used in a separate database RESET command (e.g., RESET STIFFNESS or RESET GCP_LDI). The command format is:

```
RESET LDI = ldi
```

where *ldi* is the logical device index of the database library. The default setting is:

```
RESET LDI = 1
```

7.2.6.11 The RESET MASS Command

This command is used to change the default logical device index and name of the element (consistent) or nodal (lumped) mass datasets. The command format is:

```
RESET MASS = [ ldi, ] ds_name
```

where *ldi* is the logical device index of the computational database and *ds_name* is the new dataset name. The default settings are:

```
RESET MASS = ldi, EltNam .MASS...mesh
```

for the element consistent mass matrix dataset, and:

```
RESET MASS = ldi, NODAL.MASS...mesh
```

for the lumped mass matrix dataset. The variable *ldi* is the logical device index (*ldi*) defined via the RESET LDI command, *EltNam* is the current element processor/type name defined via the RESET ELEMENT_TYPE command, and *mesh* is the mesh number defined via the RESET MESH command. For the element consistent mass name, the user does not have to type the element name, but may instead use the abbreviation E*. For example, the command:

```
RESET MASS = E*.CONSISTENT_MASS
```

would result in the dataset name being changed to *EltNam*.CONS_MASS, where *EltNam* is automatically replaced with the definition set via the RESET ELEMENT_TYPE command.

7.2.6.12 The RESET MESH Command

This command is used to change the default mesh number used in all dataset names (unless otherwise specified via a separate dataset RESET command). The command format is:

```
RESET MESH = mesh
```

where *mesh* is an integer number, typically set to the current mesh number. The default setting is:

```
RESET MESH = 0
```

which corresponds to the initial mesh. Relevant before all DEFINE and FORM commands.

7.2.6.13 The RESET NL_GEOM Command

This command is used to change the default geometric nonlinearity option. It is often used in conjunction with the RESET COROTATION command. The command format is:

RESET NL_GEOM = *nl_geom_option*

where

nl_geom_option	Description
0	The analysis is geometrically linear; linear element strain-displacement relations will be employed, and element corotation will be disregarded. (Default)
1	The analysis is geometrically nonlinear, but only linear element strain-displacement relations will be used. Geometric nonlinearity must be accounted for via element corotation (see RESET COROTATION command), which for many beam/shell element types is not as accurate as option 2.
2	The analysis is geometrically nonlinear, and nonlinear element strain-displacement relations will be used. Element corotation may or not be selected with this option. For many beam/shell element types, nonlinear element strain-displacement relations enhances corotation, making it more accurate for a given mesh and rotation magnitude.

The RESET NL_GEOM command is relevant before the following action commands: FORM STIFFNESS, FORM FORCE, FORM STRESS, FORM STRAIN, and FORM STRAIN_ENERGY.

7.2.6.14 The RESET NL_LOAD Command

This command is used to change the default load nonlinearity option. It affects whether "live" loads are to be processed as part of the external force vector, or the tangent stiffness matrix. The command format is:

RESET NL_LOAD = *nl_load_option*

where

nl_geom_load	Description
0	Ignore load nonlinearity (i.e., displacement dependence). Only displacement-independent ("dead") external loads are to be processed in the following FORM FORCE or FORM STIFFNESS command. (Default)
1	Include load nonlinearity. Only displacement-dependent ("live") external loads are to be processed in the following FORM FORCE or FORM STIFFNESS command.

The RESET NL_LOAD command is relevant before the following action commands: FORM STIFFNESS/TANG, FORM FORCE/EXT, and FORM FORCE/RES.

7.2.6.15 The RESET NL_MATL Command

This command is used to change the default material nonlinearity option. The command format is:

```
RESET NL_MATL = nl_matl_option
```

where

nl_matl_option	Description
0	The analysis is materially linear; ignore nonlinearity in any material constitutive models. (Default)
1	The analysis is materially nonlinear, include nonlinearity in material constitutive models if it exists.

The RESET NL_MATL command is relevant before the following action commands: FORM STIFFNESS, FORM FORCE/INT, FORM FORCE/RES, FORM STRESS, and FORM STRAIN_ENERGY.

7.2.6.16 The RESET PARAMETERS Command

This command is used to specify optional element research parameters, which are element-type dependent and hence described under individual ES*i* element processor sections in Chapter 7. The command format is:

```
RESET PARAMETERS = p1, p2, p3, . . .
```

where *p1, p2, p3, . . .*, denote floating-point parameters. A maximum of 10 such parameters is currently permitted. The default setting is:

```
RESET PARAMETERS = 0., 0., 0., . . .
```

The RESET PARAMETERS command is relevant only before the DEFINE ELEMENT commands, which saves these parameters in the database (i.e., the parameters cannot be redefined during the solution phase of the analysis).

7.2.6.17 The RESET PROJECTION Command

This command is used to change the default element "rigid-body projection" option. The rigid-body projection option is the linearized counterpart of the corotation option and modifies the stiffness matrix and displacement vector so that they are free from spurious strains due to (infinitesimal) rigid-body motion. This is relevant only for elements that do not preserve rigid-

body modes exactly (for example, warping-sensitive shell elements such as those in processor ES5) and can make a difference in both linear and nonlinear analysis. The command format is:

RESET PROJECTION = *projection_option*

where

projection_option	Description
0	Element rigid-body projection will not be performed. (Default)
1	Element rigid-body projection will be performed.

The RESET PROJECTION command is relevant before the following action commands: FORM STIFFNESS, FORM FORCE/INT, FORM FORCE/RES, FORM STRESS, FORM STRAIN, and FORM STRAIN_ENERGY.

7.2.6.18 The RESET ROTATION Command

This command is used to change the default logical device index and name of the nodal rotation (pseudovector) dataset. The command format is:

RESET ROTATION = [*ldi*,] *ds_name*

where *ldi* is the logical device index of the computational database and *ds_name* is the new dataset name. The default setting is:

RESET ROTATION = *ldi*, NODAL.ROTATION.*step.mesh*

where *ldi* is the logical device index defined via the RESET LDI command, *step* is the load/time-step number defined via the RESET STEP command, and *mesh* is the mesh number defined via the RESET MESH command. For example, the command:

RESET ROTATION = NODAL.PSEUDO_VECTOR.*step.*

would set the rotation dataset name to NODAL.PSEUDO_VECTOR.*step*, without including the mesh number in the name.

7.2.6.19 The RESET STEP Command

This command is used to change the default load- or time-step number used in many solution dataset names (unless otherwise specified via a separate dataset RESET command). The command format is:

RESET STEP = *step*

where *step* is an integer number, typically set to the current step number. The default setting is:

RESET STEP = 0

which corresponds to the linear (or initial) solution. If *step* = 0, then *ldset* is used in solution dataset names, as specified via the RESET LOAD_SET command. (Relevant before all FORM commands.)

7.2.6.20 The RESET STIFFNESS Command

This command is used to change the default logical device index and name of the element stiffness matrix dataset. The command format is:

RESET STIFFNESS = [*ldi*,] *ds_name*

where *ldi* is the logical device index of the computational database and *ds_name* is the new dataset name. The default setting is:

RESET STIFFNESS = *ldi*, *EltNam*.STIFFNESS...*mesh*

where *ldi* is the logical device index defined via the RESET LDI command, *EltNam* is the current element processor/type name concatenation defined via the RESET ELEMENT_TYPE command, and *mesh* is the mesh number defined via the RESET MESH command. The user does not have to type the element name, but may instead use the abbreviation E*. For example, the command:

RESET STIFFNESS = E*.MATL_STIFFNESS

would result in the dataset name being changed to *EltNam*.MATL_STIFFNESS, where *EltNam* is automatically replaced with the definition set via the RESET ELEMENT_TYPE command. The above command would typically be done before invoking the FORM STIFFNESS/MATL command.

7.2.6.21 The RESET STRAIN Command

This command is used to change the default logical device index and name of the element strain dataset before using the FORM STRAIN command. It also causes strains to be output to the database by the FORM STRESS, FORM FORCE/INT, or FORM FORCE/RES commands. The command format is:

RESET STRAIN = [*ldi*,] *ds_name*

where *ldi* is the logical device index of the computational database and *ds_name* is the new dataset name. The default setting is:

RESET STRAIN = *ldi*, *EltNam*.STRAIN.*step.mesh*

where *ldi* is the logical device index defined via the RESET LDI command, *step* is the load/time-step number defined via the RESET STEP command, *EltNam* is the current element name defined via the RESET ELEMENT_TYPE command, and *mesh* is the mesh number defined via the RESET MESH command. The user does not have to type the full element name for *EltNam*, but may abbreviate it as E*. For example, the command:

RESET STRAIN = E*.STRAIN_FAB_DIR.*step*

would set the strain dataset name to *EltNam*.STRAIN_FAB_DIR.*step*, where *EltNam* is automatically replaced for E*. This would be appropriate if the user had set the strain direction option to FAB_DIR (via the RESET STR_DIRECTION command) so that the strain components were expressed in the material fabrication coordinate system. Relevant for the following commands: FORM STRAIN, FORM STRESS, FORM FORCE/EXT, and FORM FORCE/RES.

7.2.6.22 The RESET STRAIN_ENERGY Command

This command is used to change the default logical device index and name of the element strain-energy density dataset before using the FORM STRAIN_ENERGY command. It also causes strain-energy densities to be output to the database by the FORM STRESS, FORM FORCE/RES, or FORM FORCE/INT commands. The command format is:

RESET STRAIN_ENERGY = [*ldi*,] *ds_name*

where *ldi* is the logical device index of the computational database and *ds_name* is the new dataset name. The default setting is:

RESET STRAIN_ENERGY = *ldi*, *EltNam*.STRAIN_ENERGY.*step.mesh*

where *ldi* is the logical device index defined via the RESET LDI command, *step* is the load/time-step number defined via the RESET STEP command, *EltNam* is the current element name defined

via the RESET ELEMENT_TYPE command, and *mesh* is the mesh number defined via the RESET MESH command. The user does not have to type the full element name for *EltNam*, but may abbreviate it as E*. For example, the command:

RESET STRAIN_ENERGY = E*.Uhat.*step*

would set the strain dataset name to *EltNam*.Uhat.*step*, where *EltNam* is automatically substituted for E*. Relevant for the following commands: FORM STRAIN_ENERGY FORM STRESS, FORM FORCE/EXT, and FORM FORCE/RES.

7.2.6.23 The RESET STRESS Command

This command is used to change the default logical device index and name of the element stress dataset before using the FORM STRESS command. It also causes strains to be output to the database by the FORM FORCE/INT or FORM FORCE/RES commands. The command format is:

RESET STRESS = [*ldi*,] *ds_name*

where *ldi* is the logical device index of the computational database and *ds_name* is the new dataset name. The default setting is:

RESET STRESS = *ldi*, *EltNam*.STRESS.*step.mesh*

where *ldi* is thelogical device index defined via the RESET LDI command, *step* is the load/time-step number defined via the RESET STEP command, *EltNam* is the current element name defined via the RESET ELEMENT_TYPE command, and *mesh* is the mesh number defined via the RESET MESH command. The user does not have to type the full element name for *EltNam*, but may abbreviate it as E*. For example, the command:

RESET STRAIN = E*.STRESS_FAB_DIR.*step*

would set the strain dataset name to *EltNam*.STRESS_FAB_DIR.*step*, where *EltNam* is automatically substituted for E*. This would be appropriate if the user had set the stress direction option to FAB_DIR (via the RESET STR_DIR command) so that the stress components were expressed in the material fabrication coordinate system. Relevant for the following commands: FORM STRESS, FORM FORCE/EXT, and FORM FORCE/RES.

7.2.6.24 The RESET STR_DIR Command

This command is used to change the default stress or strain reference frame option prior to use of the FORM STRAIN, FORM STRESS, FORM FORCE/INT, or FORM FORCE/RES commands. The command format is:

RESET STR_DIR = *str_direction*

where

str_direction	Description
ELEMENT	Use element local (integration point) reference frame, x_l, y_l, z_l, as stress/strain output reference frame: x_s, y_s, z_s.
GLOBAL { X | Y | Z }	The stress/strain output x_s axis is parallel to the global x_g, y_g, or z_g axis if X, Y or Z, respectively, is used in the subcommand. The stress/strain output z_s axis is parallel to the local element normal axis for shell elements, otherwise it is obtained by permuting the global axes. The stress/strain output y_s axis is defined by the right-hand rule.
FAB_DIR	Use the local material fabrication reference frame, x_f, y_f, z_f, as the stress/strain output reference frame, x_s, y_s, z_s.

The default setting is:

RESET STR_DIR = ELEMENT

If surface-oriented stress/strain output directions are desired, it may be necessary to define surface-oriented fabrication directions when the elements are defined (see the FAB_DIR subcommand under the DEFINE ELEMENTS command), and then RESET STR_DIR=FAB_DIR to employ these directions for stress/strain output. This option can be useful even for isotropic material based fabrications, where the fabrication direction is irrelevant to the constitutive model.

7.2.6.25 The RESET STR_LOC Command

This command is used to change the default stress, strain, or strain-energy location option prior to use of the FORM STRAIN, FORM STRESS, FORM STRAIN_ENERGY, FORM FORCE/INT, or FORM FORCE/RES commands. The command format is:

RESET STR_LOC = *str_location*

where

str_location	Description
INTEG_PTS	Element stresses, strains, or strain-energy densities will be evaluated and stored at element integration points in the STR attribute of the specified EST dataset.

str_location	**Description**
NODES	Element stresses, strains, or strain-energy densities will be evaluated at integration points, then extrapolated and stored at element nodes in the STRNOD attribute of the specified EST dataset.
CENTROIDS	Element stresses, strains, or strain-energy densities will first be evaluated at the element integration points, then averaged and stored at element centroids in the STRCEN attribute of the specified EST dataset. (If one of the element's integration points coincides with the centroid, the value computed there will be output rather than an average integration-point value.)

The default setting is:

```
RESET STR_LOC = INTEG_PTS
```

If element error estimation is desired, it may be necessary to output element stresses, strains, and/or strain-energy densities at element integration points. Refer to the particular Error Estimation processor section for details.

Stresses, strains, and/or strain-energy densities may be formed and stored at all three locations (INTEG_PTS, NODES, and CENTROIDS) by issuing three separate FORM commands. All three locations may be stored in the same stress, strain, or strain-energy dataset (as attributes STR, STRNOD, and STRCEN, respectively).

Finally, the NODES option does not lead to globally continuous nodal values. Different elements attached to the same node may produce different stress/strain/energy values for the corresponding element node. A nodal averaging post-processor is necessary to obtain globally continuous nodal values. Such an algorithm is used, for example, by the ARG graphical post-processor, discussed in Chapter 14.

7.2.7 ES Processor/Procedure Interface

Other than defining element connectivity, loads, and other attributes—which may be totally different for different element types participating in the same model—most element functions can be invoked with the same specifications for all element types. For example, while the DEFINE ELEMENTS command requires a list of element nodal connectivity for each element of each type, the FORM STIFFNESS command involves only some optional RESET commands that are typically the same for all element types participating in the model. Such commands may be invoked for all pertinent element processors/types via one call to the ES Utility Procedure. The general form of the procedure call is:

```
*call ES   ( FUNCTION = command_name ;
             Reset_arg_1  = reset_val_1     ; --
             Reset_arg_2  = reset_val_2     ; --
                :
             Reset_arg_n  = reset_val_n )
```

where *command_name* is the name of any valid ES command (e.g., FORM FORCE/EXT, FORM STIFFNESS/MATL) except for DEFINE ELEMENTS, DEFINE LOADS, or DEFINE ATTRIBUTES. The phrase *Reset_arg_i* (where i = 1, 2, ..., n) represents any of the RESET command names (e.g., COROTATION, NL_GEOM, STR_DIR, ...) and *reset_pars_i* denotes the associated command parameters.

For details on the use of the ES utility procedure, refer to Procedure ES in Section 5.2.

7.2.8 ES Processor Limitations

Element processor limitations are discussed according to specific element type, under the corresponding individual element processor (ES*i*) sections, later in this chapter.

7.2.9 ES Processor Error Messages

Some of the most important generic element processor error messages are summarized in Table 7.2-31.

Table 7.2-31 Summary of Error Messages Printed by Generic Element Processor (ES)

Error #	Error Message	Probable Cause(s)	Recommended User Response
1	# of elt ***** too large	Element developer error; size of an element parameter such as number of nodes exceeds ES dimension.	Increase ES dimension and regenerate the element processor.
2	Constitutive error	Error status returned to element processor by Generic Constitutive Processor.	Read GCP error message (which should follow) and refer to Error Messages section in Chapter 8.
3	Invalid command: *****	Either user entered invalid command in model definition procedure, or there is a system error in one of COMET-AR's solution procedures.	If in user-written model definition procedure, check command syntax in current Section. If in solution procedure, inform COMET-AR development staff.
4	Cannot open dataset *****	The specified dataset is probably not on the database file.	Make sure you have pointed to the right database file in your directory.
5	Variable properties not implemented	User has tried to define a model with material properties varying within individual elements.	Modify model definition so that material properties are constant within each element (they may vary from element to element).
6	Invalid element type [ES0LDN]	A new element type has been implemented that is not accommodated by the generic element processor.	Have element developer and/or COMET-AR development staff get together and modify ES.
7	Error encountered in element kernel [ES0CR]	An error deep within the element developer's code.	Contact element developer (if possible) or COMET-AR development staff.

7.2.10 ES Processor Examples and Usage Guidelines

7.2.10.1 Element Connectivity Definition Example: *Rectangular Shell-Element Mesh*

Figure 7.2-1 shows a simple rectangular mesh with four 4-node shell elements.

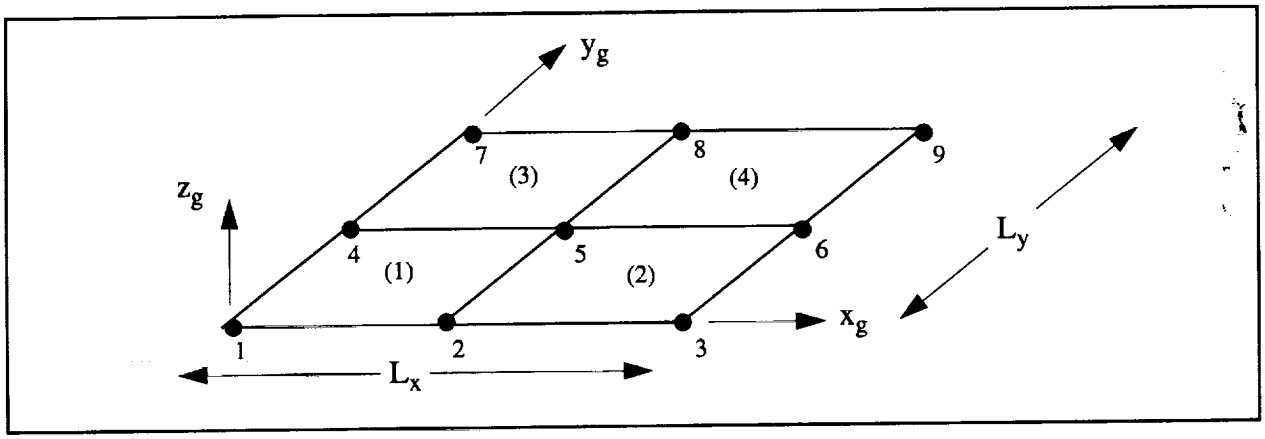


Figure 7.2-1 Rectangular Shell-Element Mesh

A sample ES command runstream to generate the above mesh is presented below.

```
RUN ES7p
        RESET ELEMENT_TYPE = SHELL
        DEFINE ELEMENTS /SOLID_MODEL=DISCRETE /P=1
          GROUP = 1
          FAB_ID = 1
          FAB_DIR = GLOBAL X
            ELEMENT = 1        NODES = 1, 2, 5, 4
            ELEMENT = 2        NODES = 2, 3, 6, 5
          GROUP = 2
            ELEMENT = 3        NODES = 4, 5, 8, 7
            ELEMENT = 4        NODES = 5, 6, 9, 8
        END_DEFINE_ELEMENTS
STOP
```

In the above example, element processor ES7p (discussed in Section 7.7) is first executed from within the COMET-AR macro-processor using the COMET-AR RUN command, and the element type name is set to SHELL via the RESET ELEMENT_TYPE command. Then, the DEFINE ELEMENTS command is entered with the DISCRETE solid model option (which happens to be the default) and with the element polynomial order set to 1 via the /P qualifier. The /P qualifier is

only necessary for element processors that have variable-order *p* capabilities, such as processor ES7p.

Next, the element group is set to 1 (via the GROUP subcommand), the first fabrication is selected (via the FAB_ID subcommand), and the fabrication x_f axis is equated to the global x_g axis (via the FAB_DIR subcommand). Then, nodal connectivity for the bottom row of elements (1 and 2) is defined and associated with group 1. Finally, the nodal connectivity for the top row of elements is defined and associated with group 2 (via the intervening GROUP subcommand). The FAB_ID and FAB_DIR settings remain intact for the elements in group 2. The NODES subcommand must be typed on the same logical command line as the ELEMENT subcommand since the two subcommands are linked; but line continuation of the ELEMENT and NODE subcommand pair on more than one physical command line may be accomplished via continuation (--) marks.

The ES END_DEFINE_ELEMENTS subcommand terminates the element definition sequence, and the COMET_AR STOP command terminates processor ES7p.

Sophisticated do-loops, conditional statements, and symbolic replacement (i.e., parametrization) may be employed in conjunction with the DEFINE ELEMENTS command by embedding the ES processor commands in a CLAMP procedure. Consult Reference [2] for details on the CLAMP language. References [3], [4], and [5] may also be helpful for background and examples.

7.2.10.2 Element Load Definition Example: *Constant Pressure on Rectangular Plate*

In this example, we shall apply a constant pressure load to the single-surface shell-element model defined in Figure 7.2-1. The command runstream is presented below.

```
RUN ES7p
            RESET ELEMENT_TYPE = SHELL
            DEFINE LOADS /PRESSURE
               LOAD = 100.
            END_DEFINE_LOADS
STOP
```

The above runstream first resets the element type (which is necessary for each execution of an element processor), and then defines a constant pressure load of 100 (force per unit area) in the positive z_g direction (which in this case is parallel to the element outward normal direction, z_l). In the absence of any ELEMENT or GROUP subcommands, the same load value is applied to all elements of all groups.

7.2.10.3 Element Load Definition Example: *Variable Pressure on Rectangular Plate*

In this example, we shall apply a piecewise constant pressure load to the single-surface shell-element model defined in Figure 7.2-1, with a different value of pressure on each element group. The command runstream is presented below.

```
RUN ES7p
            RESET ELEMENT_TYPE = SHELL
            DEFINE LOADS /PRESSURE
              GROUP = 1
                LOAD = 100.
              GROUP = 2
                LOAD = 200.
            END_DEFINE_LOADS
STOP
```

7.2.10.4 Element Load Definition Example: *Const. Line Load on Rectangular Plate Boundary*

In this example, we shall apply a constant transverse (z_g directed) line load to the boundary y_g=0 of the rectangular shell-element model defined in Figure 7.2-1. The command runstream is presented below.

```
RUN ES7p
            RESET ELEMENT_TYPE = SHELL
            DEFINE LOADS /LINE /SYSTEM=GLOBAL
              ELEMENT = 1, 2
                LINE = 1
                  LOAD = 0., 0., -100.
            END_DEFINE_LOADS
STOP
```

In the above runstream, line loads are selected via the /LINE qualifier, and the load coordinate system is set to global, via the /SYSTEM qualifier. Then, the first edge (line 1) of elements 1 and 2 are selected via the LINE and ELEMENT subcommands, respectively. Finally, the LOAD subcommand specifies that the line load vector has a magnitude of 100 (force per unit length) in the negative z_g direction. The meaning of line "1" is an element-type-dependent definition. For the quadrilateral elements within processor ES7p, line 1 is defined as the line connecting element nodes 1 and 2. Consult the appropriate ES*i* section in this manual to obtain the correct information for a particular element type.

7.2.10.5 Element Load Definition Example: *Variable Line Load on Rect. Plate Boundary*

In this example, we extend the previous example by allowing the line load to vary linearly from x_g=0 to x_g=L_x in the rectangular shell-element model (Figure 7.2-1). The command runstream is presented below.

```
RUN ES7p
            RESET ELEMENT_TYPE = SHELL
            DEFINE LOADS /LINE /SYSTEM=GLOBAL
              LINE = 1
                ELEMENT = 1
                  NODE = 1
                    LOAD = 0., 0., 0.
                  NODE = 2
                    LOAD = 0., 0., -100.
                ELEMENT = 1
                  NODE = 1
                    LOAD = 0., 0., -100.
                  NODE = 2
                    LOAD = 0., 0., -200.
            END_DEFINE_LOADS
STOP
```

In the above runstream, element line 1 is selected first; then element line nodes 1 and 2 are loaded on line 1 of elements 1 and 2. The node numbers are relative to each element line, rather than to the element nodal connectivity order. The load of –100 at global node 2 is repeated once per element, at the corresponding element node; thus, for element 1, this load is applied to element line node 2, and for element 2, the same load is applied to element line node 1. The above definition represents a load (i.e., force per unit length) that varies from in magnitude from 0 at global node 1, to 200 at global node 3; and is pointing in the negative z_g direction.

7.2.10.6 Element Solution Formation Example: *Forming External Force Vectors*

The following example illustrates how a specific ES*i* processor may be executed to form and assemble consistent external forces based on the distributed loads defined in the previous example.

```
RUN ES7p
            RESET ELEMENT_TYPE = SHELL
            RESET LOAD_FACTOR = 2.0
            FORM FORCE/EXT
STOP
```

In the above runstream, the specific element type is selected and the load factor is changed to 2.0, via RESET commands, and then the element forces are formed and assembled via the FORM FORCE command. By default, the element forces are accumulated/output into a nodal force vector dataset called NODAL.FORCE, and that the COMET-AR STOP command (or another RUN command) are necessary to ensure that the database is properly closed.

An alternative (and more convenient) way of performing the above function is through an ES procedure call, i.e.,

```
*call ES ( FUNCTION = FORM FORCE/EXT; LOAD_FACTOR = 2.0 )
```

The difference between the above procedure call and the previous command runstream is that the element type name does not have to be specified in the procedure call. The procedure instead automatically processes (i.e., performs all of the steps listed in the previous runstream) all element processors and specific element types associated with the current model, as summarized in the CSM.SUMMARY dataset.

7.2.10.7 Element Solution Formation Example: *Computing Element Stresses*

The following example is similar to the previous example on forming external forces, except that this command runstream computes element stresses after the global displacement solution has been obtained.

```
RUN ES7p
            RESET ELEMENT_TYPE = SHELL
            RESET STR_DIR = FAB_DIR
            RESET STR_LOC = CENTROIDS
            RESET STRESS = E*.STRESS_CENT
            FORM STRESS
STOP
```

In the above runstream the stress direction is set to the fabrication direction, and the stress location specification is set to element centroids. The stress dataset name is set (implicitly) to ES7p_SHELL.STRESS_CENT...1, where the E* is automatically replaced by the actual element processor and type names (concatenated with an underscore).

Alternatively, we could have performed this same function automatically for all element types defined in the current model by employing the following ES procedure call.

```
*call ES ( FUNCTION = FORM STRESS; MESH = 2; STR_DIR = FAB_DIR ; --
            STR_LOC = CENTROIDS; STRESS = E*.STRESS_CENT...2 )
```

Something like the above procedure call is built in to utility procedures such as STRESS, which in turn is called by solution procedures such as L_STATIC_1, and AR_CONTROL_1 (see Part II of this manual for more information on the procedure interface to COMET-AR).

7.2.11 References

[1] Stanley, G. M., *The Generic Structural-Element Processor Manual for the COMET Code*, NASA CR, 1990.

[2] Felippa, C. A., *The Computational Structural Mechanics Testbed Architecture: Volume II: Directives*, NASA CR-178385, 1989.

[3] Stewart, C. B., ed., *The Computational Structural Mechanics Testbed User's Manual*, NASA TM 100644, 1989.

[4] Stewart, C. B., ed., *The Computational Structural Mechanics Testbed Procedures Manual*, NASA TM 100645, 1989.

[5] Stehlin, B. P., *The COMET-AR Tutorial Manual*, NASA CR (preliminary), February 1993.

7.3 Processor ES1 (SRI and ANS Shell Elements)

7.3.1 Element Description

Processor ES1 contains various shear-deformable (C^0) quadrilateral shell elements, including displacement-based selective-reduced integrated (SRI) elements, and assumed natural-coordinate strain (ANS) shell elements. Both SRI and ANS element families include 4-node (bilinear) and 9-node (biquadratic) element geometries.

These elements are intended for modeling very thin to moderately thick shells. Both SRI and ANS formulations are designed to alleviate common shell-element pathologies such as locking, spurious mechanisms, and mesh distortion sensitivity; however, different element types within these families achieve these goals to varying extents, and none are optimal. Many of the specific element types implemented within processor ES1 are intended primarily for research-type comparisons, and not for production analyses. The only production-oriented element types in processor ES1 are the 4- and 9-node ANS elements (especially the 9-node), called EX47 and EX97, respectively. A more efficient (but occasionally more distortion-sensitive) implementation of these (and higher-order) ANS shell elements may be found in processor ES7p.

The following sections provide an overview of the various shell element types contained within processor ES1. For a more detailed theoretical description consult Reference [1].

7.3.1.1 Summary of Element Types

Currently implemented element types available within processor ES7p are summarized in Tables 7.3-1 and 7.3-2. The first contains 4-node elements; the second contains 9-node elements.

Table 7.3-1 Summary of Processor ES1 4-node Element Types

Element Type Name	Description	Status
EX41	Uniformly reduced (1-pt) integrated (URI) element; standard isoparametric Lagrange bilinear displacement interpolation.	Research
EX42	Selectively reduced integrated (SRI) element; reduced (1-pt) integration on all shear strain components; bilinear displacements.	Research
EX43	Similar to EX42, but directionally-reduced integration is used on transverse shear strains. (Very distortion sensitive)	Research
EX44	Same as EX42 except in-plane shear strains fully integrated.	Research
EX45	Same as EX43 except geometric stiffness is fully integrated.	Research
EX46	Fully integrated bilinear Lagrange element (locks in bending).	Research
EX47	Assumed natural-coordinate strain (ANS) element; bilinear geometry and displacements; constant strain field.	Production

Table 7.3-2 Summary of Processor ES1 9-node Element Types

Element Type Name	Description	Status
EX91	Uniformly reduced (2x2) integrated (URI) element; isoparametric Lagrange biquadratic displacement interpolation.	Research
EX92	Uniformly reduced integrated (URI) Serendipity element; only 8 nodes are active; the 9th (center) node is inactive.	Research (untested)
EX93	Selectively reduced integrated (SRI) Heterosis element; the 9th (center) node has active rotations, but no translations. Reduced integration on membrane and transverse-shear strains, as well as on the entire geometric stiffness matrix.	Research (untested)
EX94	Same as EX93 except geometric stiffness is fully integrated.	Research (untested)
EX95	Same as EX93 except geometric stiffness is selectively integrated.	Research (untested)
EX96	Fully integrated (3x3) biquadratic Lagrange element. (Stiff in bending when modeling curved shells.)	Res./Prod.
EX97	Assumed natural-coordinate strain (ANS) element; biquadratic geometry and displacements; linear strain field.	Production

7.3.1.2 Element Geometry and Node Numbering

The geometry, node and integration point numbering conventions for 4-node elements are illustrated in Figure 7.3-1. Similar information is provided for 9-node elements in Figure 7.3-2. In these figures, element nodes are shown as solid circles with bold node numbers, and element Gauss integration (stress storage) points for fully integrated and selectively-reduced integrated element types are shown as X's with plain number subscripts. The integration point locations for elements that employ uniform reduced integration (i.e., EX41 and EX91) are shown in part **c** of these figures as small x's. Selectively reduced integrated elements (e.g., EX42 and EX92) employ a combination of full and reduced Gauss integration point locations for strain evaluation, but always store the resulting strains and stresses at the full integration points. Element boundary (line) numbers and node numbering conventions within boundaries (for line load application) are shown in part **b** of these figures.

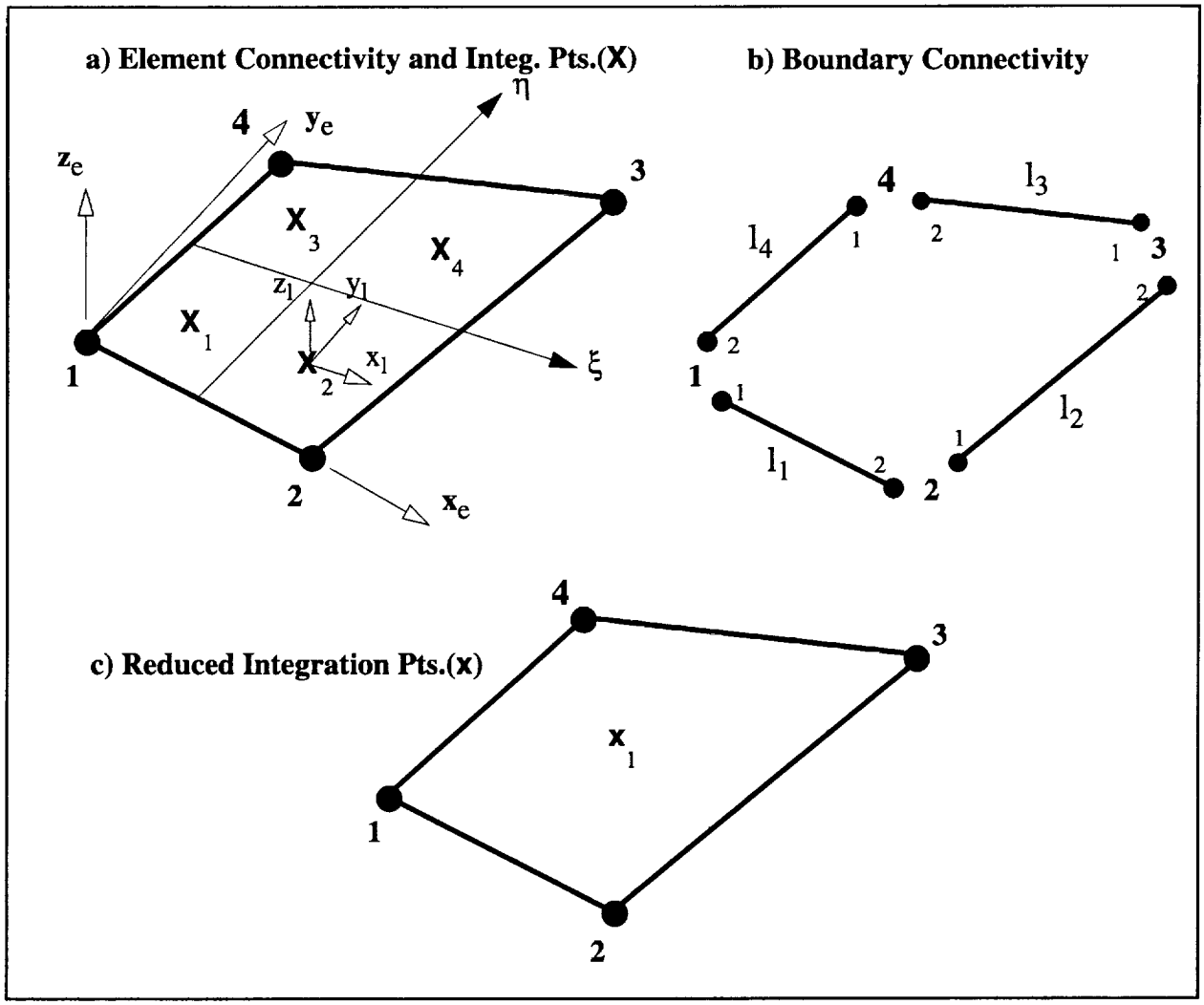


Figure 7.3-1 4-Node Element Geometry, Nodes, and Integration Points

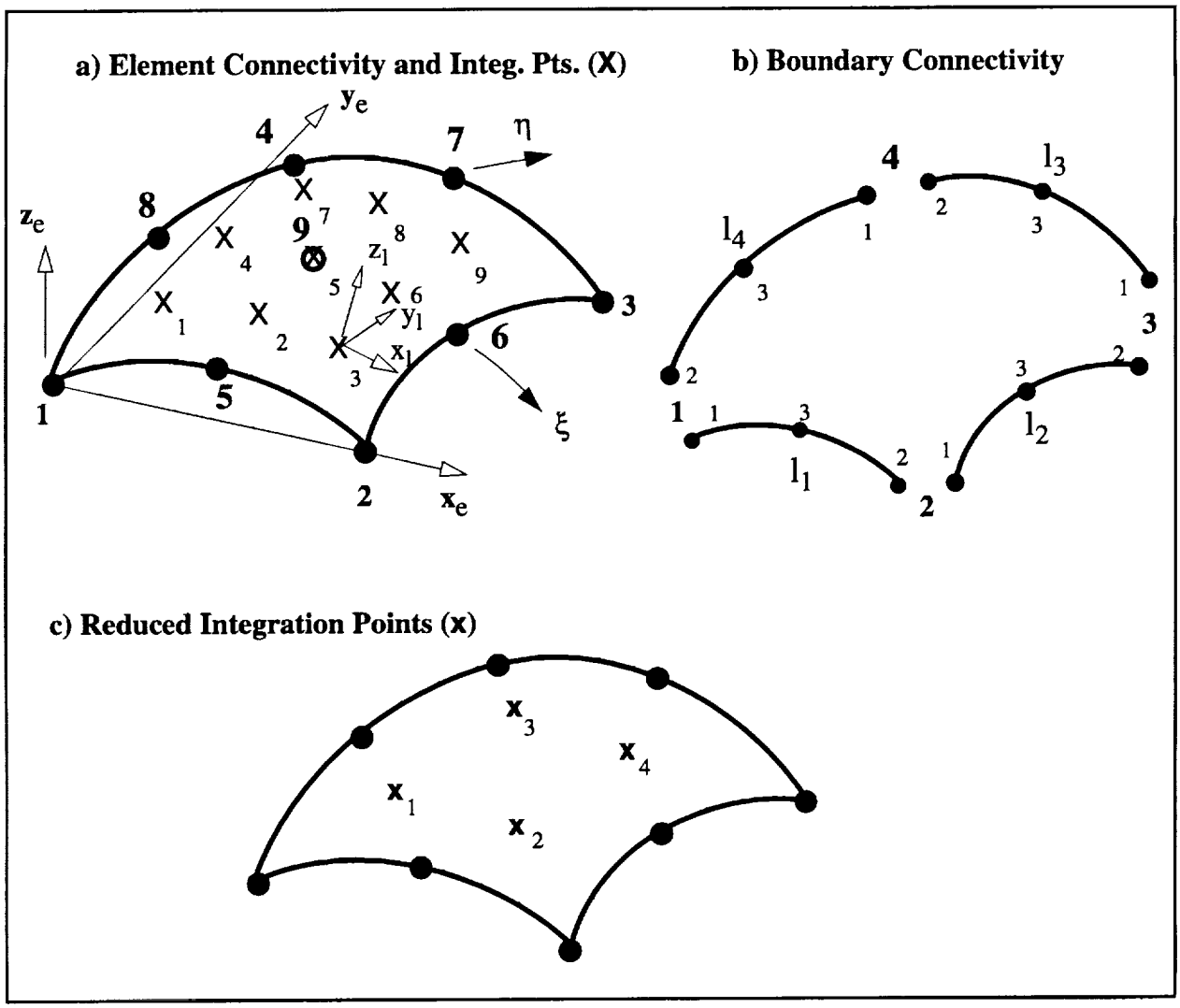


Figure 7.3-2 9-Node Element Geometry, Nodes, and Integration Points

In Figures 7.3-1 and 7.3-2, the orthogonal x_e, y_e, z_e axes form the element Cartesian (or corotational) coordinate system. The orthogonal x_l, y_l, z_l axes form the element local stress coordinate system, which can vary from integration point to integration point; and the non-orthogonal/curvilinear ξ, η, ζ axes from the element natural-coordinate system. The x_e axis initially connects nodes 1 and 2, and the z_e axis is perpendicular to the 1-2-3 plane; however, this coordinate system is slightly modified by the generic element processor to achieve a less biased system for corotational nonlinear analysis (see Reference [2]). The x_l axis is always tangent to the local ξ curve, the z_l axis is always normal to the ξ–η tangent plane, and the y_l axis completes an orthogonal triad.

7.3.1.3 Nodal Freedoms (DOFs) and BCs

All of the quadrilateral shell elements in Processor ES1 have 3 translational displacement DOFs and 3 rotational displacement DOFs at each element node (see Figure 7.3-3); however, the

"drilling" rotational DOF (i.e., the rotation about the local element surface-normal vector) does not have any intrinsic stiffness, and a "drilling stabilization" option must be employed with this element (see subsection on Drilling Stabilization later in this section).

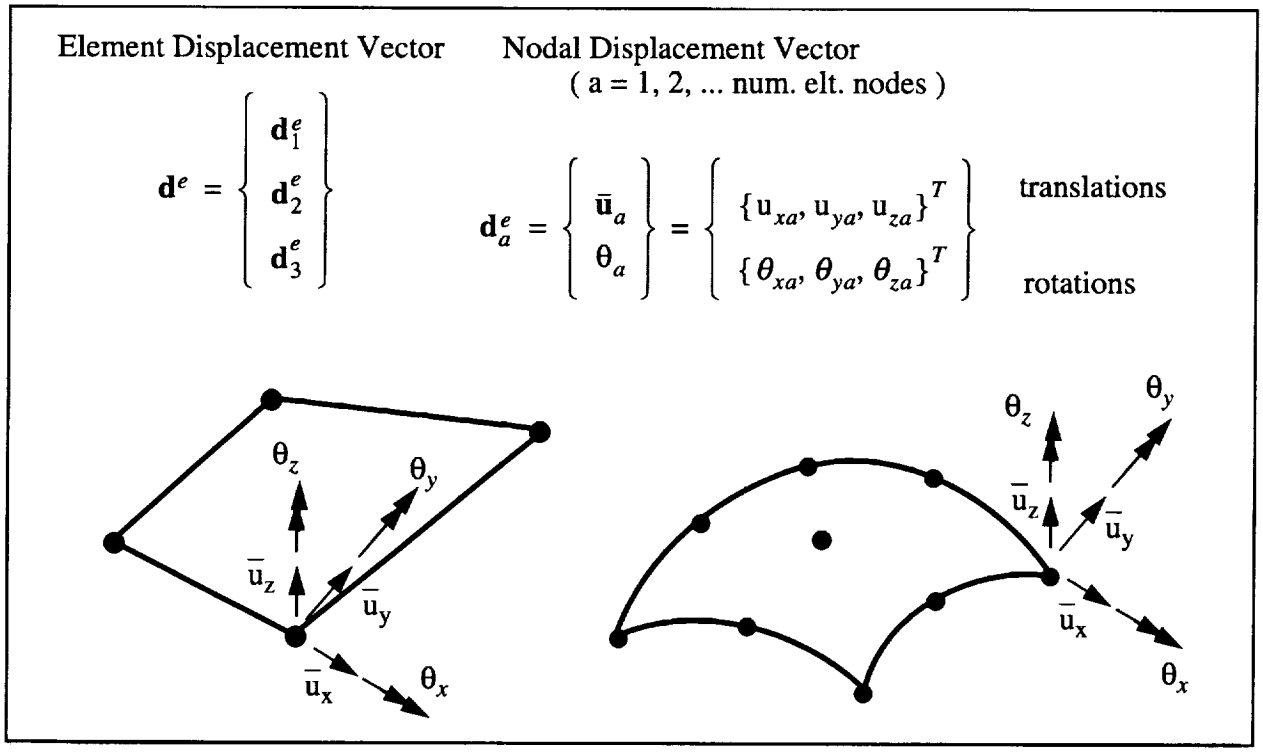


Figure 7.3-3 Displacement DOFs for ANS Shell Elements

7.3.1.4 Displacement (and Geometry) Representation

The approximation of the displacement field, as well as the surface geometry, within both SRI and ANS shell elements, is defined by Lagrangian interpolation functions, which have the variations shown in Table 7.3-3.

Table 7.3-3 Processor ES1 Shell Element Displacement and Geometry Approximations

Component	Polynomial Variation	
	4-Node Elements	9-node Elements
$\bar{u}(\xi,\eta)$, $\theta(\xi,\eta)$ and $x(\xi,\eta)$	$p_1(\xi)*p_1(\eta)$	$p_2(\xi)*p_2(\eta)$

where $p_i()$ denotes a polynomial of degree i in the argument variable. For SRI and ANS shell elements, the strains are not obtained by simply differentiating the displacement field, as explained in the following subsection.

7.3.1.5 Strain Representation

Both classes of elements generate 8 resultant strain components, which are stored at each of the element integration (i.e., "stress storage") points. The 8-strain resultants are arranged as follows:

$$\varepsilon = \begin{bmatrix} \bar{\varepsilon} \\ \kappa \\ \gamma \end{bmatrix} = \begin{bmatrix} \text{Membrane_Strains} \\ \text{Bending_Strains} \\ \text{Transverse-Shear_Strains} \end{bmatrix}$$

where

$$\bar{\varepsilon} = \begin{bmatrix} \bar{\varepsilon}_x \\ \bar{\varepsilon}_y \\ \bar{\varepsilon}_{xy} \end{bmatrix} \qquad \kappa = \begin{bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix} \qquad \gamma = \begin{bmatrix} \gamma_x \\ \gamma_y \end{bmatrix}$$

where the subscripts x and y denote the x_l and y_l components at an integration point (see Figures 7.3-1 and 7.3-2).

For the SRI elements, the Cartesian strain components are obtained by first differentiating the assumed displacement field, then sampling specific strain components at either reduced or full integration points, and finally extrapolating the sampled strains to the full integration points. The variation of the strains within an element is thus filtered by the sampling points (see [1] and [4] for details). Similarly, for the ANS elements, the strain components are sampled at reduced integration points but in a directional manner, different for each strain component, and expressed in a curvilinear (natural-coordinate) basis (see [1] and [3] for details). The resulting intra-element variations for individual strain components within each of the ES1 element types is summarized in Tables 7.3-4 and 7.3-5.

Table 7.3-4 Processor ES1 4-node Shell Element Strain Approximations

Elt. Type	Membrane Strains			Bending Strains			Transverse-Shears	
	e_x	e_h	e_{xh}	k_x	k_h	k_{xh}	g_x	g_h
SRI								
EX41	$p_0(\xi,\eta)$	$p_0(\xi,\eta)$	$p_0(\xi,\eta)$	$p_0(\xi,\eta)$	$p_0(\xi,\eta)$	$p_0(\xi,\eta)$	$p_0(\xi,\eta)$	$p_0(\xi,\eta)$
EX42	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$	$p_0(\xi,\eta)$	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$	$p_0(\xi,\eta)$	$p_0(\xi,\eta)$	$p_0(\xi,\eta)$
EX43	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$	$p_0(\xi,\eta)$	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$	$p_0(\xi,\eta)$	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$
EX44	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$	$p_1(\xi,\eta)$	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$	$p_1(\xi,\eta)$	$p_0(\xi,\eta)$	$p_0(\xi,\eta)$
EX45	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$	$p_0(\xi,\eta)$	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$	$p_0(\xi,\eta)$	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$
EX46	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$	$p_1(\xi,\eta)$	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$	$p_1(\xi,\eta)$	$p_1(\xi,\eta)$	$p_1(\xi,\eta)$
ANS								
EX47	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$	$p_0(\xi,\eta)$	$p_0(\xi)*p_1(\eta)$	$p_1(\xi)*p_0(\eta)$	$p_0(\xi,\eta)$	$p_0(\xi,\eta)$	$p_0(\xi,\eta)$

Table 7.3-5 Processor ES1 9-node Shell Element Strain Approximations

Elt. Type	Membrane Strains			Bending Strains			Transverse-Shears	
	e_x	e_h	e_{xh}	k_x	k_h	k_{xh}	g_x	g_h
SRI								
EX91	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$
EX92	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$
EX93	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_2(\eta)$	$p_2(\xi)*p_1(\eta)$	$p_2(\xi,\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$
EX94	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_2(\eta)$	$p_2(\xi)*p_1(\eta)$	$p_2(\xi,\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$
EX95	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_2(\eta)$	$p_2(\xi)*p_1(\eta)$	$p_2(\xi,\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$
EX96	$p_1(\xi)*p_2(\eta)$	$p_2(\xi)*p_0(\eta)$	$p_2(\xi,\eta)$	$p_1(\xi)*p_2(\eta)$	$p_2(\xi)*p_1(\eta)$	$p_2(\xi,\eta)$	$p_2(\xi,\eta)$	$p_2(\xi,\eta)$
ANS								
EX97	$p_1(\xi)*p_2(\eta)$	$p_2(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_2(\eta)$	$p_2(\xi)*p_1(\eta)$	$p_1(\xi)*p_1(\eta)$	$p_1(\xi)*p_2(\eta)$	$p_2(\xi)*p_1(\eta)$

7.3.1.6 Stress Representation

Stress resultants conjugate to the above strain resultants are computed via the Generic Constitutive Processor (see Chapter 8), and are arranged as follows.

$$\sigma = \begin{bmatrix} \mathbf{N} \\ \mathbf{M} \\ \mathbf{Q} \end{bmatrix} = \begin{bmatrix} \text{Membrane_Stresses} \\ \text{Bending_Stresses} \\ \text{Transverse-Shear_Stresses} \end{bmatrix}$$

where

$$\mathbf{N} = \begin{bmatrix} N_x \\ N_y \\ N_{xy} \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} M_x \\ M_y \\ M_{xy} \end{bmatrix} \qquad \mathbf{Q} = \begin{bmatrix} Q_x \\ Q_y \end{bmatrix}$$

Like the strains, the stress resultants are also computed and stored at the element integration (i.e., stress storage) points, and have the same polynomial variations (for linear constitutive models).

7.3.1.7 Drilling Rotational Stiffness

Since the present shell element formulation has no intrinsic drilling (normal rotational) stiffness, an artificial drilling stiffness option is provided. This option is triggered by the AUTO_DRILL solution procedure argument and works as shown in Figure 7.3-4.

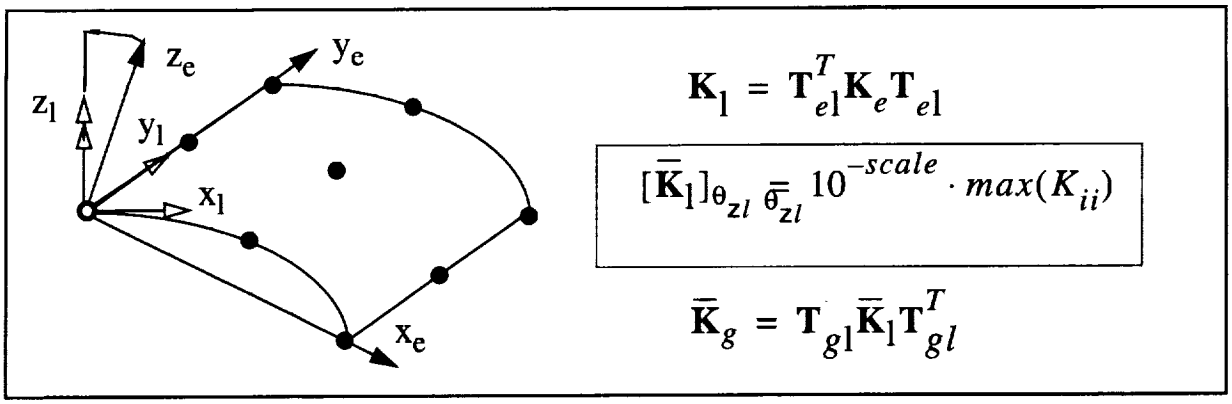


Figure 7.3-4 Implementation of Artificial Drilling Stiffness in Processor ES1

The element material stiffness matrix is first computed in the element corotational frame (x_e,y_e,z_e) and then rotated into an independent local frame (l) at each node such that the zl axis is parallel to the element normal (or drilling) axis. The diagonal drilling rotational stiffness components are then set equal to a small fraction of the maximum element diagonal stiffness component. Finally, the element matrix is rotated back to the element corotational frame before depositing in the database for assembly. The fractional coefficient multiplying the maximum diagonal stiffness component involves a negative power of 10. That exponent, referred to as *scale*, corresponds to the *scale* parameter in the AUTO_DRILL solution procedure argument. The default coefficient is 10^{-5} (*scale*=5).

7.3.1.8 Element Nonlinearity

Element geometrical nonlinearity is accounted for by an Updated Lagrangian treatment of the element force vector and stiffness matrix, and by a moderate-rotation nonlinear strain measure based on the midpoint strain tensor (see [4] for details). Additionally, both SRI and ANS shell elements may be (and should be) employed with the generic element processors (ES) built-in corotational capability to enable arbitrarily large rotations (see [2] for details). For material nonlinearity, Processor ES1 is fully compatible with the generic constitutive processor, and all specific shell constitutive models implemented therein (see Chapter 8).

7.3.2 Element Command Specifications

General command syntax and options are all inherited from the generic element processor (see Section 7.2). Special command options for Processor ES1 are described in the following subsections.

7.3.2.1 RESET Command for Element Type

The various element types within Processor ES1 can be selected via the command:

```
RESET ELEMENT_TYPE = EltTyp
```

where, for example, *EltTyp* would be EX47 for the 4-ANS element and EX97 for the 9-ANS element. This RESET command should be entered before using the DEFINE ELEMENTS command. ES1 is not a "p" type element processor, so the /P qualifier should not be used with the DEFINE ELEMENTS command.

7.3.2.2 RESET Command for Element-Specific Research Parameters

None.

7.3.2.3 RESET Commands for Drilling Stiffness and Angle Tolerance

The default *scale* parameter used to compute artificial drilling stiffness is 5, which corresponds to a scale factor of 10-5 (see Figure 7.3-4). The value of *scale* can be changed via the command:

```
RESET DRILL_STIFF = scale
```

The default angle tolerance for requiring artificial drilling stiffness is 1 degree. Drilling stiffness flags are turned on at any node for which the normals of all attached shell elements make an angle less than this tolerance with the average element normal. The default tolerance can be changed via the command:

```
RESET DRILL_TOL = angle
```

Both of the above parameters also appear in the AUTO_DRILL solution procedure argument, and the angle tolerance parameter appears in the AUTO_MPC, AUTO_TRIAD, and AUTO_DOF_SUP solution procedure arguments.

7.3.3 Element Input/Output Datasets

General input and output dataset specifications are inherited from the generic element processor (see Section 7.2). There are no special-purpose datasets or data attributes at this time.

7.3.3.1 Auxililiary Storage Dataset

None.

7.3.3.2 Other Special-Purpose Datasets/Attributes

None.

7.3.4 Element Implementation Status and Limitations

A summary of the current implementation status of the shell elements within processor ES1 is given in Table 7.3-6. All functions except for the load stiffness matrix and element-dependent error estimates are implemented for all element types. Neither of these two functions is essential. Generic element error estimates are adequate for adaptive refinement, and the load stiffness matrix is important only for some buckling eigenproblems involving live loads (e.g., hydrostatically loaded cylindrical shells).

Table 7.3-6 Processor ES1 Shell Element Implementation Status

Functions	SRI Elements	ANS Elements
Auto DOF Suppression	Yes	Yes
Body Forces	Yes	Yes
Consistent Mass	Yes	Yes
Diagonal Mass	Yes	Yes
Drilling Stiffness	Yes	Yes
Error Estimates/Elt-dep.	No	No
Error Estimates/Generic	Yes	Yes
Geometric Nonlinearity	Yes	Yes
Geometric Stiffness	Yes	Yes
Internal Forces	Yes	Yes
Load Stiffness	No	No
Material Nonlinearity	Yes	Yes
Pressure Forces	Yes	Yes
Strains	Yes	Yes
Stresses	Yes	Yes
Stress Extrapolation	Yes	Yes
Stress Transformation	Yes	Yes
Surface Forces	Yes	Yes

7.3.5 Element Error Messages

A summary of the most important or common error messages associated specifically with processor ES7p are described in Table 7.3-7.

Table 7.3-7 Summary of Element Processor ES7p Error Messages

Error #	Error Message	Probable Cause(s)	Recommended User Response
1	Invalid ES1 element type	The user has selected an invalid element type (via the RESET ELEMENT_TYPE command) when defining element connectivity or loads.	Change the element type to one of the valid names listed in Tables 1 and 2.
2	ES0**** not implemented	The element developer has not implemented this particular element function.	Try to work around the unimplemented function; or ask the element developer to implement it ASAP.
3	Zero determinant of Jacobian	The element nodes probably do not define a proper quadrilateral. Either the nodal coordinates are not as intended by the user, or the definition of element nodal connectivity via the DEFINE ELEMENTS command is incorrect.	Check nodal coordinates and element connectivity. (This error is probably *not* due to the degeneration of a quadrilateral into a triangle; that is a permissible modeling technique with this element processor.)

7.3.6 Element Selection and Usage Guidelines

The following element selection and usage guidelines are based on experience to-date with COMET-AR Processor ES1.

7.3.6.1 Element Type Selection

As indicated under the general Element Description subsection, only three of the fourteen shell element types implemented in processor ES1 are recommended for general-purpose analysis (in order of preference):

1) EX97 (9-node ANS element)

2) EX96 (9-node fully integrated Lagrange element)

3) EX47 (4-node ANS element)

The EX97 (or 9ANS) element is the most robust of the element types; the EX96 (or 9LAG) element is robust, but can be excessively stiff relative to EX97 for curved shell models. The EX47 (or 4ANS) is also robust, but not as efficient as the 9ANS element, and provides a much cruder representation of the geometry of curved shells (e.g., 10 EX47 elements over a 90-degree circular arc are equivalent to about 3 EX97 elements for geometric accuracy).

> ***Caution:*** The remaining element types are included only for research purposes, serving as benchmarks for comparison with newer element formulations. Some of these research elements may exhibit pathologies such as spurious modes, or locking phenomena, which are not considered safe for production-oriented analysis.

7.3.6.2 Problem Class Recommendations

The shell elements in Processor ES1 are all equipped for general-purpose linear/nonlinear/static/dynamic structural analysis; however, for nonlinear analysis, while rotations can be arbitrarily large (with the corotational option), strains are assumed to be relatively small (<5%).

7.3.6.3 Distortion Sensitivity

While the ANS shell elements (EX47 and EX97) are in general more accurate than the SRI elements (e.g., EX42 and EX96) for a given mesh, the ANS elements tend to be more distortion sensitive. That is, as the element shapes in the mesh are made to deviate more and more from rectangular, the solution degrades faster for ANS elements than for many of the SRI elements. If mesh distortion is kept within reasonable limits and not allowed to increase as the mesh is refined, the ANS elements should converge to the exact solution more rapidly than the others. All of the quadrilateral elements within processor ES1 can be degenerated into triangular elements by allowing all of the nodes on one side to correspond to a single global node. The degenerated triangular elements exhibit a degradation in accuracy that is greater for the 4-node elements than for the 9-node elements, and also greater for the ANS element than for the SRI elements in general.

7.3.6.4 Automatic Drilling Stabilization

Because neither the ANS nor the SRI shell elements within processor ES1 have intrinsic drilling rotational stiffness, the user must select one of the automatic drilling DOF stabilization options available in COMET-AR solution procedures (see Section 2.10): either the AUTO_DRILL option (which will engender artificial drilling stiffness at the element level); or the AUTO_DOF_SUP option (which will suppress global rotational DOFs if the computational axes are closely aligned with the element normal). The AUTO_TRIAD option may also be selected in conjunction with the AUTO_DOF_SUP option, if the computational axes are not closely aligned with the element normals. Finally, an AUTO_MPC option, which automatically generates an explicit multi-DOF constraint to eliminate the drilling rotation at appropriate nodes, is also available at the solution procedure level.

At shell/shell, or shell/stiffener junctures, drilling stabilization is unnecessary.

7.3.6.5 Adaptive Analysis Guidelines

All of the shell elements in Processor ES1 may be used in conjunction with adaptive mesh refinement (AR) with the following provisos.

1) The EX47 (4-ANS) and EX97 (9-ANS) shell elements can be distortion-sensitive when used with transition-based (h_t) refinement; if h_t refinement is to be used, the EX97 element is recommended over the EX47 element.

2) The EX47 and EX97 elements are also sensitive to the multipoint constraints generated by constraint-based (h_c) refinement; again, the EX97 is recommended over the EX47.

3) The only SRI element that is appropriate for both h_t and h_c adaptive mesh refinement is the EX96 (fully integrated 9-LAG) element.

7.3.7 References

[1] Stanley, G. M., *The Computational Structural Mechanics Testbed [COMET] Structural Element Processor ES1: Basic SRI and ANS Shell Elements*, NASA CR 4357, 1990.

[2] Stanley, G. M., *The Generic Structural-Element Processor Manual for the COMET Code*, NASA CR 181728, 1990.

[3] Park, K. C. and Stanley, G. M., "A Curved C0 Shell Element Based on Assumed Natural Coordinate Strains," *Journal of Applied Mechanics*, Vol. 108, pp. 278-290, 1986.

[4] Stanley, G. M., "Continuum-Based Shell Elements," Ph.D. Thesis, Stanford University, Stanford, CA, 1985.

7.4 Processor ES5 (STAGS Shell Element)

7.4.1 Element Description

Processor ES5 contains a flat 4-node Kirchhoff-type (transverse-shear-free) shell element with intrinsic drilling stiffness. This element was transferred directly from the NASA-sponsored STAGS finite-element code [1], where it has been used as a "work-horse" element for over a decade, and the implementation in COMET-AR should be identical to that found in STAGS. Within STAGS the element is known as the 410 shell element; the element type name in COMET-AR has been changed to E410.

The E410 shell element is recommended only for thin shells (with negligible transverse-shear flexibility) and in conjunction with fairly rectangular element shapes. Element accuracy tends to degenerate rapidly with either in-plane mesh distortion or out-of-plane warping; however, the element's intrinsic drilling stiffness is a real advantage, making it unnecessary to use any of the automatic drilling DOF suppression options implemented in COMET-AR (e.g., AUTO_MPC). For a more detailed theoretical description of the E410 element, see Reference [2].

7.4.1.1 Summary of Element Types

There is currently only one element type available within processor ES5, as summarized in Table 7.4-1.

Table 7.4-1 Summary of Processor ES5 Element Types

Element Type Name	Description	Status
E410	4-node flat Kirchhoff-type shell element with drilling stiffness	Implemented

7.4.1.2 Element Geometry and Node Numbering

The E410 shell element geometry and node numbering is illustrated in Figure 7.4-1. Element nodes are shown as solid circles with bold node numbers, and integration (stress-storage) points are shown as X's with plain number subscripts. Element boundary (line) numbers and node numbering conventions within boundaries (for line load application) are shown in part **b** of the figure.

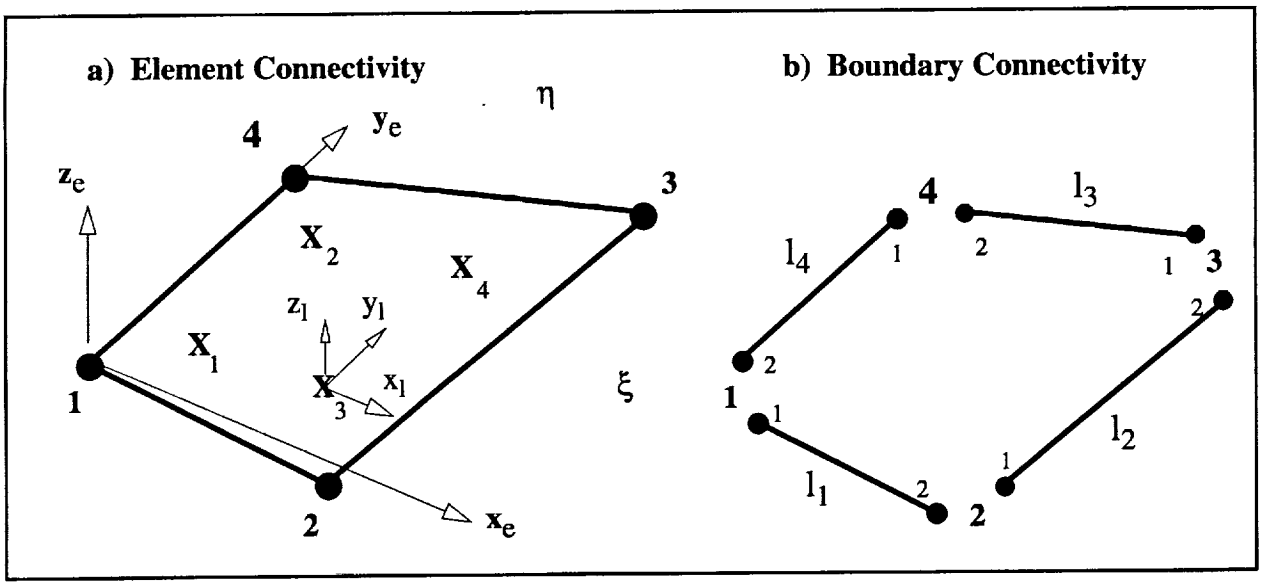


Figure 7.4-1 E410 Shell Element Geometry and Node Numbers

In Figure 7.4-1, the element corotational frame (x_e, y_e, z_e) is defined such that, for flat elements, the y_e axis is parallel to the side connecting element nodes 1 and 4 (i.e., l_4), the z_e axis is normal to lines l_1 and l_4, and the x_e axis completes an orthogonal triad. For "warped" elements, this preliminary frame is rotated so that the z_e axis is normal to the plane defined by the two element diagonals (1-3 and 2-4).

The integration point x_l, y_l, z_l axes are parallel to the x_e, y_e, z_e axes, and hence fixed throughout the element. The E410 element is always formulated as a flat element, with nodes of the actual (user-specified) element geometry projected onto the flat surface defined by the average normal vector, z_e. The 4-point Gauss integration rule used corresponds to slightly reduced integration of the element stiffness matrix and internal force vectors (which require a 5-point rule for exact integration). This rule improves element performance without introducing spurious kinematic modes (at least in all element test cases run to-date).

7.4.1.3 Nodal Freedoms (DOFs) and BCs

The E410 quadrilateral shell element in Processor ES5 has 3 translational displacement DOFs and 3 rotational displacement DOFs at each element node (see Figure 7.4-2). All 3 rotational DOFs have intrinsic stiffness associated with them, even the so-called drilling rotations (i.e., the rotations about the z_e axis at each node), which are linked to the membrane strain field.

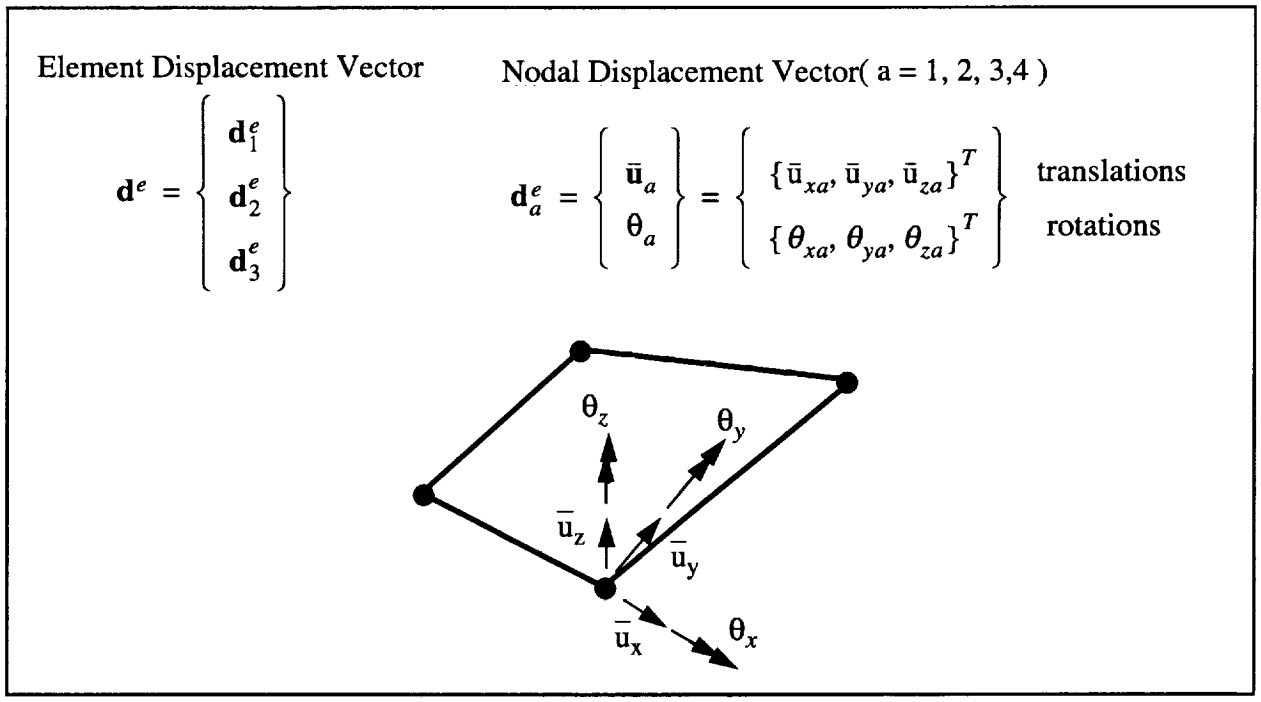


Figure 7.4-2 Displacement DOFs for E410 Shell Element

7.4.1.4 Displacement Representation

The approximation of the displacement field within the E410 shell element is based on a nonconforming cubic polynomial for the bending (transverse) displacement component (w), and a combination of cubic and linear polynomials for the in-plane displacement components (u, v). This is summarized in Table 7.4-2.

Table 7.4-2 Processor ES5, Element E410 Displacement Approximations

Component	Approximation
Transverse: $w = \bar{u}_{ze}(x_e, y_e)$	Cubic (nonconforming) polynomial; function of nodal values of transverse displacement and rotations: $\bar{u}^a_{ze}$, θ^a_{xe} and θ^a_{ye}.
In-Plane: $u = \bar{u}_{xe}(\xi_e, \psi_e)$, $v = \bar{u}_{ye}(\xi_e, \psi_e)$	Mixed cubic/linear polynomial; function of nodal values of in-plane translations and drilling rotations: $\bar{u}^a_{xe}$, $\bar{v}^a_{xe}$, θ^a_{ze}.

7.4.1.5 Strain Representation

The E410 shell elements in Processor ES5 generates 6 resultant strain components, which are stored at each of the element's 4 integration points. The 6-strain resultants are arranged as follows:

$$\varepsilon = \begin{bmatrix} \bar{\varepsilon} \\ \kappa \end{bmatrix} = \begin{bmatrix} \text{Membrane_Strains} \\ \text{Bending_Strains} \end{bmatrix}$$

where

$$\bar{\varepsilon} = \begin{bmatrix} \bar{\varepsilon}_x \\ \bar{\varepsilon}_y \\ \bar{\varepsilon}_{xy} \end{bmatrix} \qquad \kappa = \begin{bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix}$$

where the "e" subscript has been dropped for simplicity.

These strain components are constructed within each element domain by differentiating the displacement approximation (Table 7.4-2), using the standard strain-displacement definitions given in Table 7.4-1, for linear analysis. The resulting variations of each strain component in the element x, y directions is also shown in Table 7.4-3.

Table 7.4-3 Element E410 Strain Definitions

Strain Component	Definition in Terms of Displacement Components	Polynomial Variation due to Displacement Approximation
ε_x	$u_{,x}$	$p_0(x_e) \times p_2(y_e)$
ε_y	$v_{,y}$	$p_2(x_e) \times p_0(y_e)$
ε_{xy}	$u_{,y} + v_{,x}$	$p_2(x_e,y_e)$
κ_x	$w_{,xx}$	$p_1(x_e,y_e)$
κ_y	$w_{,yy}$	$p_1(x_e,y_e)$
κ_{xy}	$-2w_{,xy}$	$p_1(x_e,y_e)$

where $p_i(x)$ refers to a polynomial of degree "i" in the x direction.

7.4.1.6 Stress Representation

Stress resultants conjugate to the above strain resultants are computed via the Generic Constitutive Processor (GCP), and are arranged as follows:

$$\sigma = \begin{bmatrix} \mathbf{N} \\ \mathbf{M} \end{bmatrix} = \begin{bmatrix} \text{Membrane_Stresses} \\ \text{Bending_Stresses} \end{bmatrix}$$

where

$$\mathbf{N} = \begin{bmatrix} N_x \\ N_y \\ N_{xy} \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} M_x \\ M_y \\ M_{xy} \end{bmatrix}$$

Like the strains, the stress resultants are also computed and stored at the element integration points, and have the same polynomial variations (for linear constitutive models).

7.4.1.7 Drilling Rotational Stiffness

As mentioned above, the E410 element has intrinsic drilling rotational stiffness which emanates from its membrane strain field. No special measures have to be taken to suppress drilling rotational DOFs.

7.4.1.8 Element Nonlinearity

Element geometrical nonlinearity is accounted for by a Total Lagrangian treatment of the element force vector and stiffness matrix; and by a moderate-rotation nonlinear strain measure based on the Lagrangian strain tensor. It is recommended that the user employ the standard COMET-AR corotational option (see COROTATION argument in analysis procedures such as AR_CONTROL and NL_STATIC_1) in conjunction with the E410 shell element. This will refer the Total Lagrangian formulation to an element corotational frame and enable arbitrarily large rotations (albeit only small to moderate strains). For material nonlinearity, Processor ES5 is fully compatible with the Generic Constitutive Processor (GCP), and all specific shell constitutive models implemented therein.

7.4.2 Element Command Specifications

General command syntax and options are all inherited from the generic element processor (see Section 7.2). Special command options for Processor ES5 are described in the following subsections.

7.4.2.1 RESET Command for Element Type

While there is only one element type E410 within processor ES5, the user must explicitly define the element type via the command:

```
RESET ELEMENT_TYPE = E410
```

before using the DEFINE ELEMENTS command.

7.4.2.2 RESET Command for Element-Specific Research Parameters

None.

7.4.2.3 RESET Commands for Drilling Stiffness and Angle Tolerance

None (element E410 in processor ES5 has intrinsic drilling stiffness).

7.4.3 Element Input/Output Datasets

General input and output dataset specifications are inherited from the generic element processor (see Section 7.2). Any special-purpose datasets or data attributes are discussed in the following subsections.

7.4.3.1 Auxiliary Storage Dataset

Processor ES5 creates an auxiliary storage dataset, called ES5_E410.AUX_STORAGE, during the initialization phase of analysis. This dataset contains pre-computed element kinematic data that is employed repeatedly during the course of an analysis.

7.4.3.2 Other Special-Purpose Datasets/Attributes

None.

7.4.4 Element Implementation Status and Limitations

A summary of the current implementation status of the E410 shell element within processor ES5 is given in Table 7.4-4.

Table 7.4-4 Processor ES5, Shell Element E410 Implementation Status

Functions	Status
Auto DOF Suppression	N/A
Body Forces	No
Consistent Mass	No
Diagonal Mass	No
Drilling Stiffness	Yes
Error Estimates/Elt-dep.	No
Error Estimates/Generic	Yes
Geometric Nonlinearity	Yes
Geometric Stiffness	Yes
Internal Forces	Yes
Load Stiffness	No
Material Nonlinearity	Yes (GCP)
Material (Linear) Stiffness	Yes
Pressure Forces	Yes
Strains	Yes
Stresses	Yes (GCP)
Stress Extrapolation	Yes
Stress Transformation	Yes
Surface Forces	No

7.4.5 Element Error Messages

A summary of the most important, or most common, error messages associated specifically with processor ES5 are described in Table 7.4-5.

Table 7.4-5 Summary of Element Processor ES5 Error Messages

Error #	Error Message	Probable Cause(s)	Recommended User Response
1	DETERMINANT OF JACOBIAN = ***. PROBLEM TERMINATED.	The Jacobian, which is related to the element area, is non-positive. This usually occurs when either the element nodes are not numbered properly, or the nodal coordinates are incorrect.	Check the element nodal connectivity and nodal coordinates to make sure the element geometry is a proper quadrilateral, with no re-entrant corners.

Table 7.4-5 Summary of Element Processor ES5 Error Messages (Continued)

Error #	Error Message	Probable Cause(s)	Recommended User Response
2	ELEMENT IS SINGULAR	The element shape function matrix is singular; typically for the same reason(s) as in Error 1.	Same as for Error 1.

7.4.6 Element Selection and Usage Guidelines

The following element selection and usage guidelines are based on experience to-date with the E410 shell element in COMET-AR's Processor ES5, and related experience with the 410 element in the STAGS code.

7.4.6.1 Element Type Selection

There is only one element type to select: E410.

7.4.6.2 Problem Class Recommendations

The E410 element in Processor ES5 is suitable for general-purpose linear/nonlinear analysis.

7.4.6.3 Distortion Sensitivity

The E410 element is fairly distortion sensitive. It is recommended that element corner angles be in the range of 45–135 degrees, with 90 degrees being optimal. While out-of-plane distortion (i.e., warp) is compensated for by a rigid-body projection operator, accuracy may degrade with increasing warp in the initial (undeformed) mesh.

7.4.6.4 Adaptive Analysis Guidelines

The E410 element in Processor ES5 may be used in conjunction with adaptive mesh refinement (AR) with either transition-based (h_t) refinement, or constraint-based (h_c) refinement; however, better results are often obtained with h_c refinement, as less mesh distortion will be engendered.

7.4.7 References

[1] Almroth, B. O., Brogan, F. A., and Stanley, G. M., *Structural Analysis of General Shells, Vol. II: User Instructions for the STAGSC-1 Computer Code*, Report No. LMSC-D633873, Lockheed Palo Alto Research Laboratory, Palo Alto, CA, December 1982.

[2] Rankin, C. and Brogan, F., *The Computational Structural Mechanics (CSM) Testbed Element Processor ES5: STAGS Shell Element*, NASA CR 4358, 1991.

7.5 Processor ES6 (STAGS Beam Element)

7.5.1 Element Description

Processor ES6 contains a general-purpose, assumed-displacement 2-node (straight) beam element, based on Bernoulli-Euler beam theory. This element is intended for modeling slender beams (i.e., without transverse-shear flexibility) which appear in either frame structures or as stiffening elements in shell structures. It may be used to obtain a faceted model of a curved beam. The beam element includes stretching, bending, and twisting deformations, for which it employs linear, cubic, and linear displacement variations, respectively, within each element domain.

The element type name for the general beam element in processor ES6 is E210. This element was transferred directly from the STAGS finite element code [1], where it is referred to as the 210 element. The E210 beam element in processor ES6 is compatible with the E410 shell element in processor ES5 (which also was transferred from the STAGS code). For a more detailed theoretical description of the E210 element, see Reference [2].

7.5.1.1 Summary of Element Types

There is currently only one element type available within processor ES6, shown in Table 7.5-1.

Table 7.5-1 Summary of Processor ES6 Element Types

Element Type Name	Description	Status
E210	2-node straight Bernoulli-Euler beam shell.	Implemented

7.5.1.2 Element Geometry and Node Numbering

The E210 beam element geometry and node numbering is illustrated in Figure 7.5-1. Element nodes are shown as solid circles with bold node numbers, and integration (stress-storage) points are shown as X's with plain number subscripts.

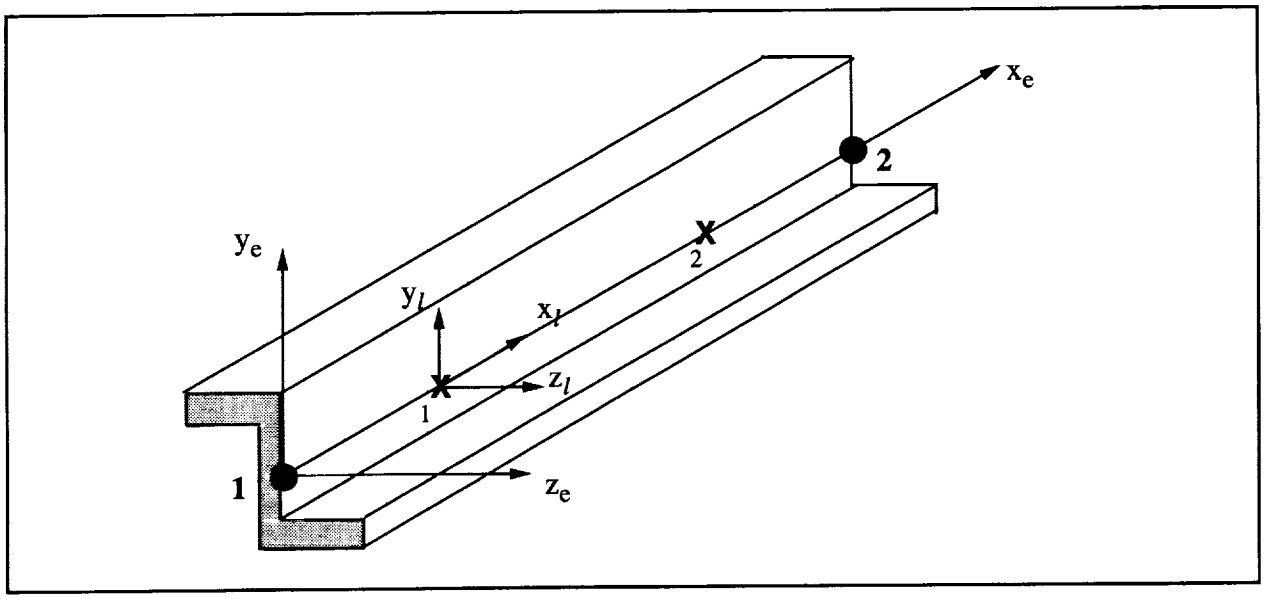


Figure 7.5-1 E210 Beam Element Geometry and Node Numbers

In Figure 7.5-1, the element corotational frame (x_e,y_e,z_e) should be defined such that the x_e axis is parallel to the line connecting nodes 1 and 2. The orientation of these corotational triads must be defined via processor TAB, and selected for individual elements within processor ES6 via the ORIENTATION subcommand of the DEFINE ELEMENTS command.

The integration point stress-storage (x_l, y_l, z_l) axes are parallel to the x_e, y_e, z_e axes, and hence fixed throughout the element. The 2-point Gauss integration rule used corresponds to uniform reduced integration of the element stiffness matrix and internal force vector, which improves element performance without introducing spurious kinematic modes.

Regarding cross-section geometry, properties such as area and moments of inertia are defined as fabrication properties via the generic constitutive processor (GCP), and the fabrication number is selected for each beam element within processor ES6 via the FABRICATION subcommand of the DEFINE ELEMENTS command. Cross-section eccentricity with respect to the nodal reference axis (1-2) can also be defined within the DEFINE ELEMENTS command, via the ECCENTRICITY subcommand.

7.5.1.3 Nodal Freedoms (DOFs) and BCs

The E210 beam element in Processor ES6 has 3 translational displacement DOFs and 3 rotational displacement DOFs at each element node (see Figure 7.5-2). While the computational directions at each node are arbitrary, the figure shows the nodal DOFs aligned with the x_e, y_e, z_e frame for convenience. Thus, in this figure, u refers to the axial displacement, v and w to the transverse (bending) displacements, θ_x refers to the torsional rotation, and θ_y and θ_z to the bending rotations.

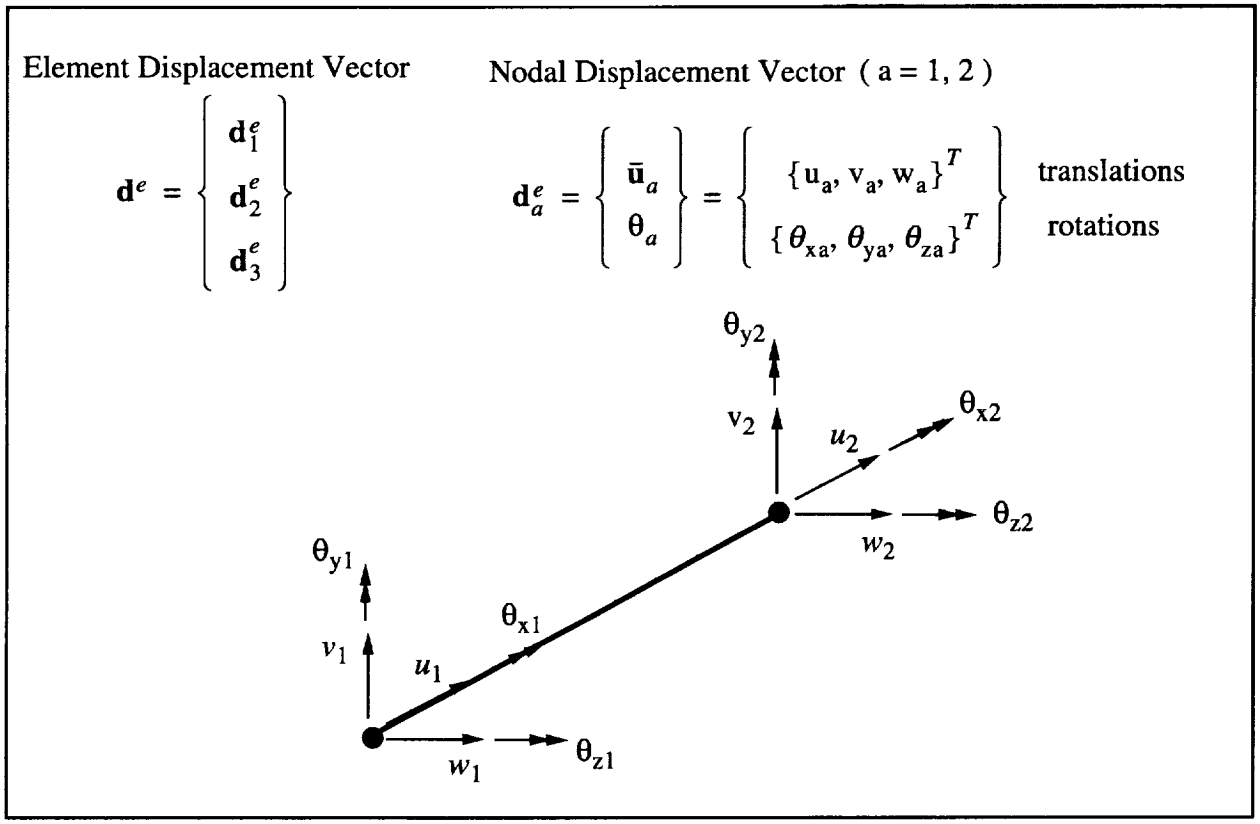


Figure 7.5-2 Displacement DOFs for E210 Shell Element

7.5.1.4 Displacement Representation

The approximation of the displacement field within the E210 beam element is based on a Hermitian cubic interpolating polynomial for the bending field, and linear interpolation for the axial and torsional displacement fields, as summarized in Table 7.5-2.

Table 7.5-2 Processor ES6, Element E210 Displacement Approximations

Component	Approximation
Axial u(x)	Linearly interpolated from nodal displacements u_1 and u_2.
Bending w(x)	Cubically interpolated, via Hermitian shape functions, from nodal displacements w_1, θ_{y1}, w_2 and θ_{z2}, where θ_y is associated with $w_{,x}$ and θ_z is associated with $v_{,x}$.
Torsion $\theta_x(x)$	Linearly interpolated from nodal rotations θ_{x1} and θ_{x2}.

In Table 7.5-2 all displacement and coordinate components are expressed in the element corotational (x_e,y_e,z_e) frame.

7.5.1.5 Strain Representation

The E210 beam element in Processor ES6 generates 4 resultant strain components, which are stored at each of the element's 2 Gauss integration points (see Figure 7.5-1). The 4-strain resultants are arranged as follows:

$$\varepsilon = \begin{bmatrix} \bar{\varepsilon}_x \\ \kappa_y \\ \kappa_z \\ \alpha \end{bmatrix} = \begin{bmatrix} \text{Axial} \\ \text{Bending_about_z} \\ \text{Bending_about_y} \\ \text{Twist} \end{bmatrix}$$

where the "e" subscript on the coordinate axes has been dropped for simplicity.

These strain components are constructed within each element domain by differentiating the displacement approximation (Table 7.5-2), using the standard strain-displacement definitions for linear analysis (see Sect. 7.5.1.7 for nonlinear analysis). The resulting variation of each strain component in the element x_e direction is shown in Table 7.5-3.

Table 7.5-3 Element E210 Strain Definitions

Strain Component	Strain Type	Definition in Terms of Displacements	Polynomial Variation within Element
$\bar{\varepsilon}_x$	Axial	$u_{,x}$	$p_0(x)$
κ_y	Bending	$-w_{,xx}$	$p_1(x)$
κ_z	Bending	$-v_{,xx}$	$p_1(x)$
a	Twist	$\theta_{x,x}$	$p_0(x)$

where commas denote differentiation and $p_i(x)$ refers to a polynomial of degree "i" in the element x_e direction.

7.5.1.6 Stress Representation

Stress resultants conjugate to the strain resultants defined in the previous section are computed via the Generic Constitutive Processor (GCP), and are arranged as follows:

$$\sigma = \begin{vmatrix} N \\ M_y \\ M_y \\ T \end{vmatrix} = \begin{bmatrix} \text{Axial_Force} \\ \text{Bending_Moment_about_z} \\ \text{Bending_Moment_about_y} \\ \text{Torque} \end{bmatrix}$$

Like the strains, the stress resultants are also computed and stored at the element integration points, and have the same polynomial variations (for linear constitutive models).

7.5.1.7 Element Nonlinearity

Element geometrical nonlinearity is accounted for by a Total Lagrangian treatment of the element force vector and stiffness matrix, and by a moderate-rotation nonlinear strain measure based on the Lagrangian strain tensor. With this strain measure, nonlinear terms are added only to the axial strain, $\bar{\varepsilon}_x$; the bending and torsional strain expressions remain linear.

It is also recommended that the user employ the standard COMET-AR corotational option (see COROTATION argument in analysis procedures such as AR_CONTROL and NL_STATIC_1) in conjunction with the E210 beam element. This will refer the Total Lagrangian formulation to an element corotational frame, and enable arbitrarily large rotations (albeit only small to moderate strains). For material nonlinearity, Processor ES6 is fully compatible with the Generic Constitutive Processor (GCP), and all specific beam constitutive models implemented therein.

7.5.2 Element Command Specifications

General command syntax and options are all inherited from the generic element processor (see Section 7.2). Special command options for Processor ES6 are described in the following subsections.

7.5.2.1 RESET Command for Element Type

While there is only one element type E210 within processor ES6, the user must explicitly define the element type via the command:

```
RESET ELEMENT_TYPE = E210
```

before using the DEFINE ELEMENTS command.

7.5.2.2 RESET Command for Element-Specific Research Parameters

None.

7.5.3 Element Input/Output Datasets

General input and output dataset specifications are inherited from the generic element processor (see Section 7.2). There are no special-purpose datasets or data attributes at this time.

7.5.3.1 Auxilliary Storage Dataset

None.

7.5.3.2 Other Special-Purpose Datasets/Attributes

None.

7.5.4 Element Implementation Status and Limitations

A summary of the current implementation status of the E210 beam element within processor ES6 is given in Table 7.5-4.

Table 7.5-4 Processor ES6, Beam Element E210 Implementation Status

Functions	Status
Auto DOF Suppression	Yes
Body Forces	No
Consistent Mass	Yes
Diagonal Mass	Yes
Error Estimates/Elt-dep.	No
Error Estimates/Generic	N/A
Geometric Nonlinearity	Yes
Geometric Stiffness	Yes
Internal Forces	Yes
Line-load Forces	Yes
Load Stiffness	No
Material Nonlinearity	Yes (GCP)
Material (Linear) Stiffness	Yes
Strains	Yes
Stresses	Yes (GCP)
Stress Extrapolation	Yes
Stress Transformation	N/A
Surface Forces	No

7.5.5 Element Error Messages

None.

7.5.6 Element Selection and Usage Guidelines

The following element selection and usage guidelines are based on experience to-date with the E210 beam element in COMET-AR's Processor ES6, and related experience with the 210 element in the STAGS code.

7.5.6.1 Element Type Selection

There is only one element type to select: E210.

7.5.6.2 Problem Class Recommendations

The E210 beam element in Processor ES6 is suitable for general-purpose linear/nonlinear/static/dynamic analysis.

7.5.6.3 Adaptive Analysis Guidelines

COMET-AR adaptive mesh refinement is not currently implemented for beam elements. Once appropriate modifications have been made to COMET-AR mesh refinement and error estimation processors (e.g., REF1 and ERRi, respectively), no specific changes should be necessary to the E210 beam element within processor ES6 to enable adaptive analysis.

7.5.7 References

[1] Almroth, B. O., Brogan, F. A., and Stanley, G. M., *Structural Analysis of General Shells, Vol. II: User Instructions for the STAGSC-1 Computer Code*, Report No. LMSC-D633873, Lockheed Palo Alto Research Laboratory, Palo Alto, CA, December 1982.

[2] Nour-Omid, S., Brogan, F. A., and Stanley, G. M., *The Computational Structural Mechanics (CSM) Testbed Element Processor ES6: STAGS Beam Element*, NASA CR 4359, 1991.

7.6 Processor ES1p (Variable-p Lagrange Quadrilateral Shell Elements)

7.6.1 Element Description

Processor ES1p contains a family of variable-polynomial(p)-order assumed displacement Lagrange (LAG) quadrilateral shell elements, ranging from a 4-node element (p=1) to a 16-node element (p=3). The formulation is based on the basic isoparametric (degenerated solid approach) described in [1], which features a C^0 (shear-deformable) shell theory [2] with Lagrange polynomial shape functions used to approximate both the element geometry and displacement field; and strains obtained by simple differentiation of the displacements. The result is a set of relatively stiff, but fairly distortion-insensitive elements. While these elements do not perform as well (i.e., converge as fast) as their high-performance counterparts (the assumed natural strain (ANS) shell elements implemented in Processor ES7p), they are often less sensitive to the side effects of adaptive mesh refinement, such as mesh distortion (in transition-based, h_t, refinement) or multi-point interelement constraints (in constraint-based, h_c, refinement).

7.6.1.1 Summary of Element Types

Currently implemented element types available within processor ES1p are summarized in Table 7.6-1.

Table 7.6-1 Summary of Processor ES1p Element Types

Element Type Name	Description	Status
SHELL (p=1)	4-node LAG (4-LAG) quadrilateral shell element; bilinear geometry and displacement field; differentiated strain field; uses selective-reduced numerical integration for shear-strain terms in element stiffness and force.	Implemented
SHELL (p=2)	9-node LAG (9-LAG) quadrilateral shell element; biquadratic geometry and displacement field; differentiated strain field; uses full numerical integration for element stiffness and force.	Implemented
SHELL (p=3)	16-node LAG (16-LAG) quadrilateral shell element; bicubic geometry and displacement field; differentiated strain field; uses full numerical integration for element stiffness and force.	Implemented

7.6.1.2 Element Geometry and Node Numbering

The three LAG shell element types listed in Table 7.6-1 are illustrated in Figures 7.6-1 to 7.6-3. Element nodes are shown as solid circles with bold node numbers, and integration (stress-storage) points are shown as X's with plain number subscripts. Element boundary (line) numbers and node numbering conventions within boundaries (for line load application) are shown in part **b** of each figure.

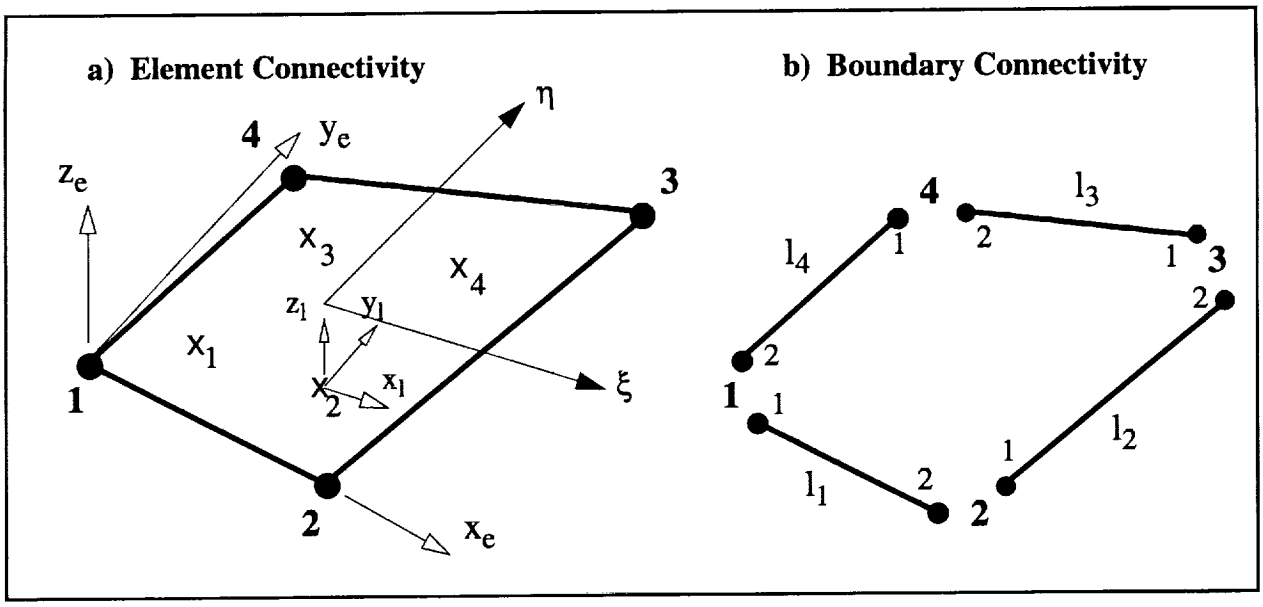


Figure 7.6-1 4-LAG (p=1) Element Geometry and Node Numbers

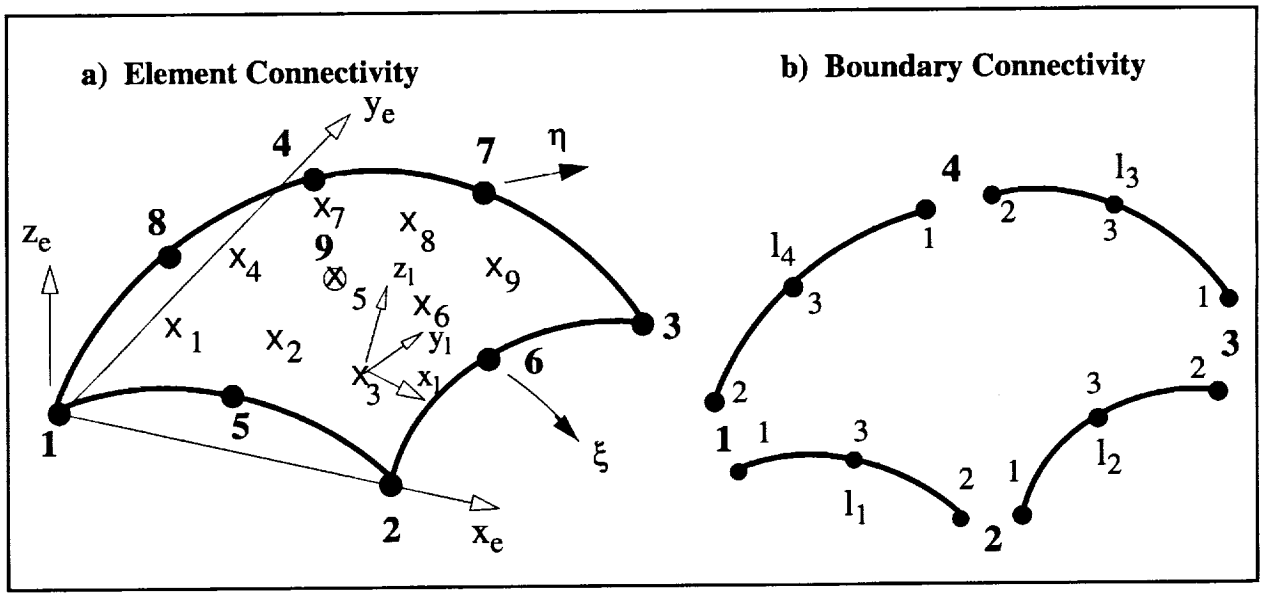


Figure 7.6-2 9-LAG (p=2) Element Geometry and Node Numbers

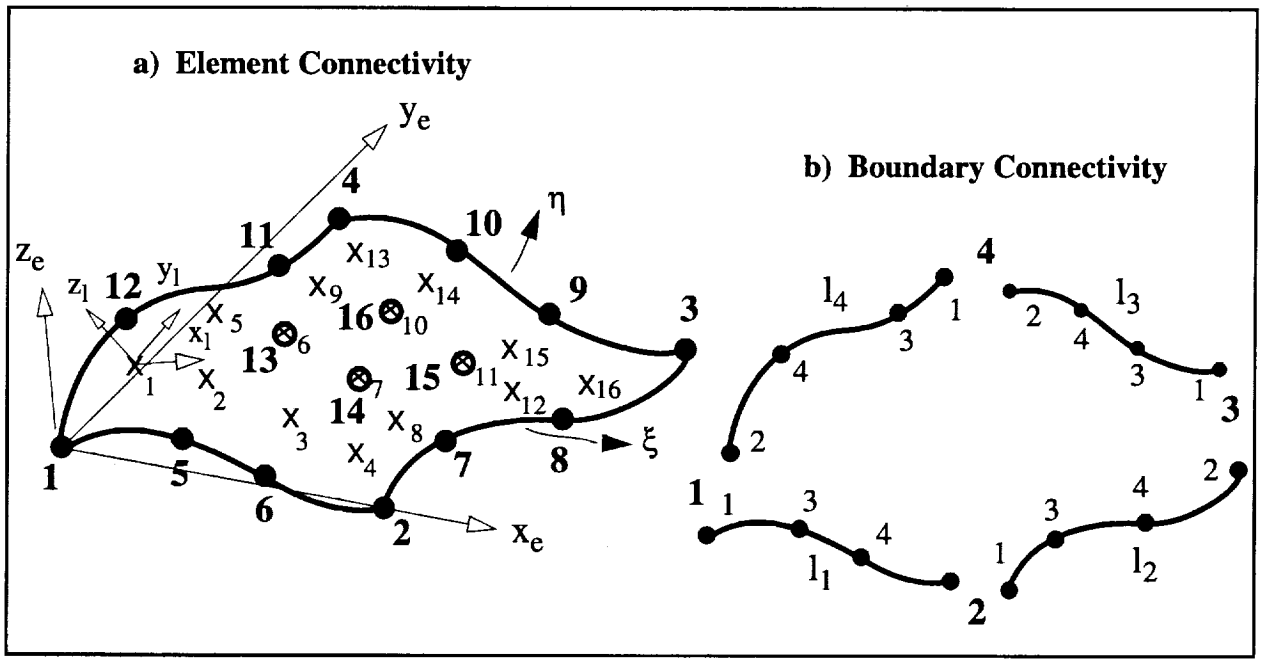


Figure 7.6-3 16-LAG (p=3) Element Geometry and Node Numbers

In Figures 7.6-1 to 7.6-3, the orthogonal x_e, y_e, z_e axes form the element Cartesian (or corotational) coordinate system; orthogonal x_l, y_l, z_l axes form the element local stress coordinate system, which can vary from integration point to integration point; and the non-orthogonal/curvilinear ξ, η, ζ axes from the element natural-coordinate system. The x_e axis initially connects nodes 1 and 2, and the z_e axis is perpendicular to the 1-2-3 plane; however, this coordinate system is slightly modified by the generic element processor to achieve a less biased system for corotational nonlinear analysis (see Reference [3]). The x_l axis is always tangent to the local ξ curve, the z_l axis is always normal to the ξ–η tangent plane, and the y_l axis completes an orthogonal triad.

7.6.1.3 Nodal Freedoms (DOFs) and BCs

All of the quadrilateral shell elements in Processor ES1p have 3 translational displacement DOFs and 3 rotational displacement DOFs at each element node (see Figure 7.6-4); however, the drilling rotational DOF (i.e., the rotation about the local element surface-normal vector) does not have any intrinsic stiffness. One of two drilling stabilization options must be employed with this element: i) artificial drilling stiffness (which may be triggered via the AUTO_DRILL option at the solution procedure level); or ii) automatic drilling DOF suppression via the AUTO_DOF_SUP option in conjunction with the AUTO_TRIAD option for models (see Section 2.10, *Automatic DOF Suppression and Drilling Stabilization*).

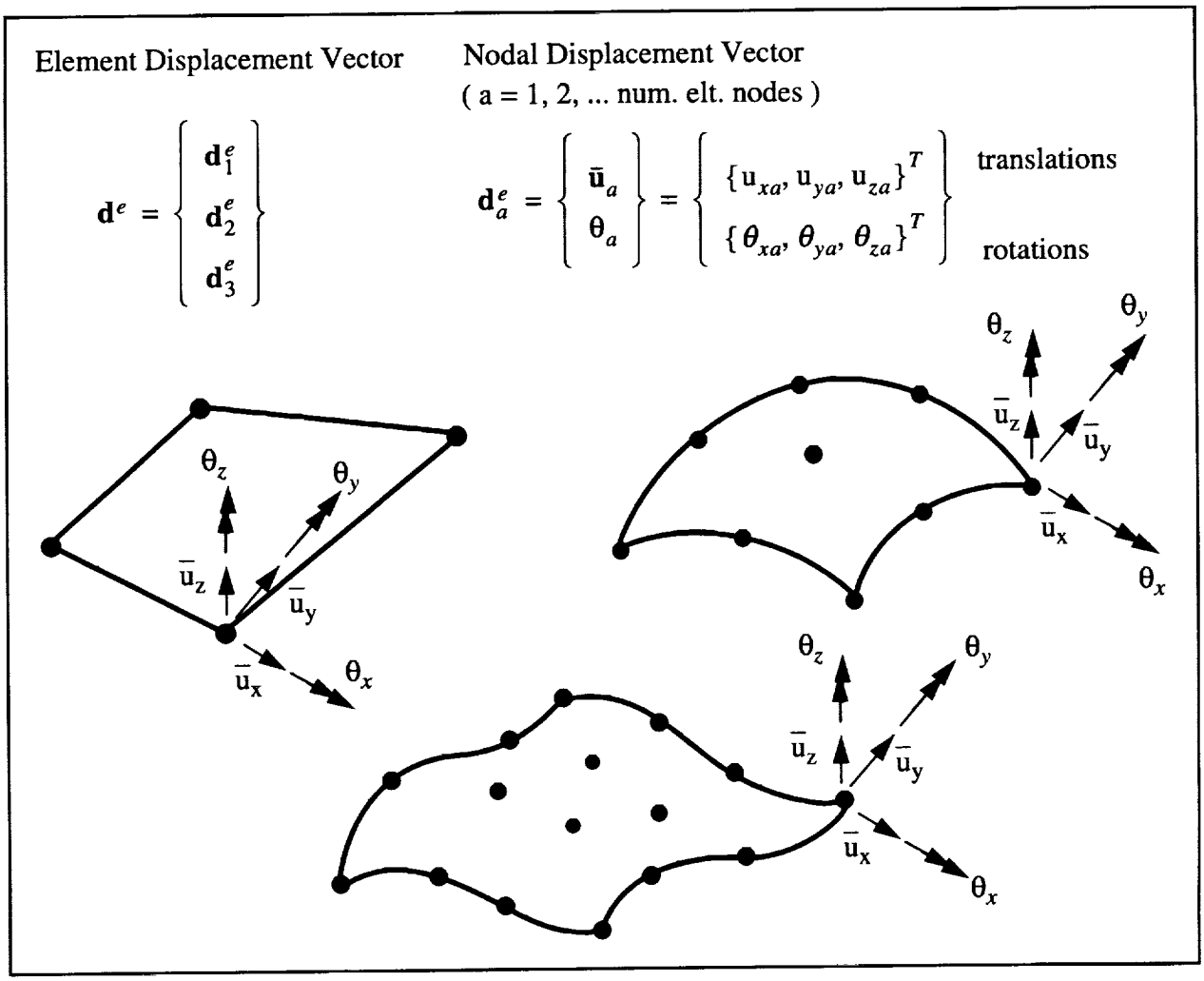


Figure 7.6-4 Displacement DOFs for LAG Shell Elements

7.6.1.4 Displacement Representation

The approximation of the displacement field within the LAG shell elements is based on Lagrange interpolating polynomials, with polynomial variations in ξ and η as shown in Table 7.6-2

Table 7.6-2 Processor ES1p Element Displacement Approximations

Component	Polynomial Variation		
	p = 1 (4-node)	p = 2 (9-node)	p = 3 (16-node)
$\bar{u}(\xi,\eta)$	Lin(ξ)*Lin(η)	Quad(ξ)*Quad(η)	Cubic(ξ)*Cubic(η)
$\theta(\xi,\eta)$	Lin(ξ)*Lin(η)	Quad(ξ)*Quad(η)	Cubic(ξ)*Cubic(η)

7.6.1.5 Strain Representation

The LAG shell elements in Processor ES1p generate 8 resultant strain components, which are stored at each of the element integration points. The 8-strain resultants are arranged as follows:

$$\varepsilon = \begin{bmatrix} \bar{\varepsilon} \\ \kappa \\ \gamma \end{bmatrix} = \begin{bmatrix} \text{Membrane_Strains} \\ \text{Bending_Strains} \\ \text{Transverse-Shear_Strains} \end{bmatrix}$$

where

$$\bar{\varepsilon} = \begin{bmatrix} \bar{\varepsilon}_x \\ \bar{\varepsilon}_y \\ \bar{\varepsilon}_{xy} \end{bmatrix} \qquad \kappa = \begin{bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix} \qquad \gamma = \begin{bmatrix} \gamma_x \\ \gamma_y \end{bmatrix}$$

where the subscripts x and y denote the x_l and y_l components at an integration point (see Figures 7.6-1 to 7.6-3). The natural-coordinate components of these strains within the element domain are obtained by differentiating the displacement approximation (Table 7.6-2), using the strain-displacement definitions given in Table 7.6-3. The 4-LAG element is a special case in that after differentiating the displacement, selective/reduced numerical integration is used on the shear-strain terms ($\varepsilon_{\xi\eta}$, $\kappa_{\xi\eta}$, γ_ξ and γ_η) appearing in the element stiffness matrix and internal force vector, such that these particular strain components are forced to remain constant throughout the element. While this improves element performance, it also engenders two spurious kinematic modes that can be triggered by certain boundary conditions (see Subsection 7.8.6 on Element Selection and Usage Guidelines).

Table 7.6-3 Processor ES1p Strain Definitions

Strain Component	Definition in Terms of Displacement Components
ε_ξ	$\frac{\partial}{\partial\xi}\bar{\mathbf{x}} \bullet \frac{\partial}{\partial\xi}\bar{\mathbf{u}}$
ε_η	$\frac{\partial}{\partial\eta}\bar{\mathbf{x}} \bullet \frac{\partial}{\partial\eta}\bar{\mathbf{u}}$
$\bar{\varepsilon}_{\xi\eta}$	$\left(\frac{\partial}{\partial\xi}\bar{\mathbf{x}} \bullet \frac{\partial}{\partial\eta}\bar{\mathbf{u}}\right) + \left(\frac{\partial}{\partial\eta}\bar{\mathbf{x}} \bullet \frac{\partial}{\partial\xi}\bar{\mathbf{u}}\right)$
κ_ξ	$\frac{\partial}{\partial\xi}\bar{\mathbf{x}} \bullet \frac{\partial}{\partial\xi}\hat{\mathbf{u}}$
κ_η	$\frac{\partial}{\partial\eta}\bar{\mathbf{x}} \bullet \frac{\partial}{\partial\eta}\hat{\mathbf{u}}$
$\kappa_{\xi\eta}$	$\left(\frac{\partial}{\partial\xi}\bar{\mathbf{x}} \bullet \frac{\partial}{\partial\eta}\hat{\mathbf{u}}\right) + \left(\frac{\partial}{\partial\eta}\bar{\mathbf{x}} \bullet \frac{\partial}{\partial\xi}\hat{\mathbf{u}}\right)$

Table 7.6-3 Processor ES1p Strain Definitions (Continued)

Strain Component	Definition in Terms of Displacement Components
γ_ξ	$\left(\hat{\mathbf{x}} \bullet \frac{\partial}{\partial \xi}\bar{\mathbf{u}}\right) + \left(\frac{\partial}{\partial \xi}\bar{\mathbf{x}} \bullet \hat{\mathbf{u}}\right)$
γ_η	$\left(\hat{\mathbf{x}} \bullet \frac{\partial}{\partial \eta}\bar{\mathbf{u}}\right) + \left(\frac{\partial}{\partial \eta}\bar{\mathbf{x}} \bullet \hat{\mathbf{u}}\right)$

The variation of the element translation vector, $\bar{\mathbf{u}}$, appearing in Table 7.6-3 is also given in Table 7.6-2. The reference-surface position vector, $\bar{\mathbf{x}}$, varies in the same way as $\bar{\mathbf{u}}$; the reference-surface normal vector is defined as the cross-product of the two in-plane tangent vectors, i.e.,

$$\hat{\mathbf{x}} = \frac{\partial}{\partial \xi}\bar{\mathbf{x}} \times \frac{\partial}{\partial \eta}\bar{\mathbf{x}}$$

and the linearized rotation vector, θ, appears implicitly through the definition of a relative displacement vector, i.e.,

$$\hat{\mathbf{u}} = -\hat{\mathbf{x}} \times \theta$$

where $\hat{\mathbf{u}}$ is the displacement of the tip of the element unit normal vector relative to the reference surface, at the point $\bar{\mathbf{x}}$.

7.6.1.6 Stress Representation

Stress resultants conjugate to the above strain resultants are computed via the Generic Constitutive Processor (GCP), and are arranged as follows:

$$\sigma = \begin{bmatrix} \mathbf{N} \\ \mathbf{M} \\ \mathbf{Q} \end{bmatrix} = \begin{bmatrix} \text{Membrane_Stresses} \\ \text{Bending_Stresses} \\ \text{Transverse-Shear_Stresses} \end{bmatrix}$$

where

$$\mathbf{N} = \begin{bmatrix} N_x \\ N_y \\ N_{xy} \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} M_x \\ M_y \\ M_{xy} \end{bmatrix} \qquad \mathbf{Q} = \begin{bmatrix} Q_x \\ Q_y \end{bmatrix}$$

Like the strains, the stress resultants are also computed and stored at the element integration points, and have the same polynomial variations (for linear constitutive models).

7.6.1.7 Drilling Rotational Stiffness

Since the present shell element formulation has no intrinsic drilling (normal rotational) stiffness, an artificial drilling stiffness option is provided. This option is triggered by the AUTO_DRILL solution procedure argument, and works as shown in Figure 7.6-5.

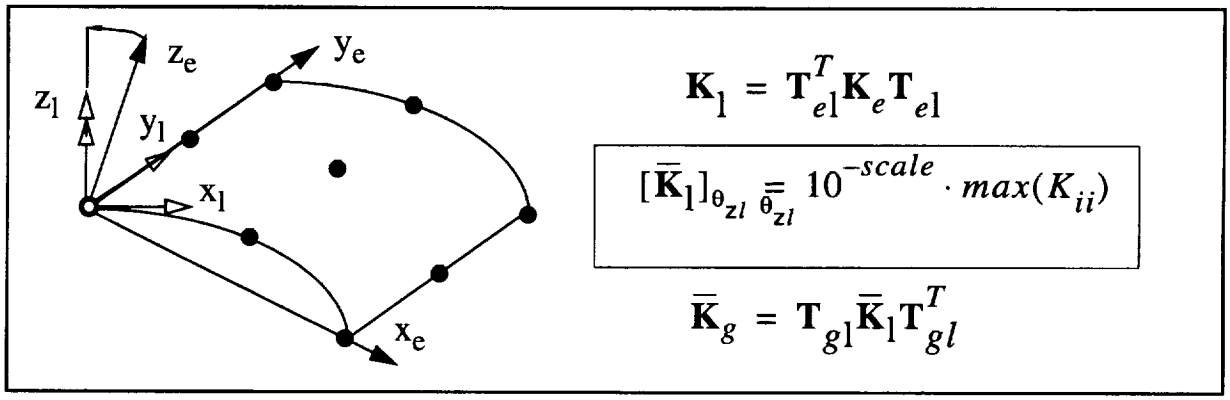


Figure 7.6-5 Implementation of Artificial Drilling Stiffness in Processor ES1p

The element material stiffness matrix is first computed in the element corotational frame (x_e,y_e,z_e) and then rotated into an independent local frame (l) at each node such that the z_l axis is parallel to the element normal (or drilling) axis. The diagonal drilling rotational stiffness components are then set equal to a small fraction of the maximum element diagonal stiffness component. Finally, the element matrix is rotated back to the element corotational frame before depositing in the database for assembly. The fractional coefficient multiplying the maximum diagonal stiffness component involves a negative power of 10. That exponent, referred to as *scale*, corresponds to the *scale* parameter in the AUTO_DRILL solution procedure argument (and also in the element processor's RESET DRILL_STIFF command). The default coefficient is 10^{-5} (*scale*=5).

7.6.1.8 Element Nonlinearity

Element geometrical nonlinearity is accounted for by an Updated Lagrangian treatment of the element force vector and stiffness matrix; and by a moderate-rotation nonlinear strain measure based on the midpoint strain tensor. Additionally, the ANS shell elements may be employed with the generic element processors (ES) built-in corotational capability to enable arbitrarily large rotations. For material nonlinearity, Processor ES1p is fully compatible with the Generic Constitutive Processor (GCP), and all specific shell constitutive models implemented therein.

7.6.2 Element Command Specifications

General command syntax and options are all inherited from the generic element processor (see Section 7.2). Special command options for Processor ES1p are described in the following subsections.

7.6.2.1 RESET Command for Element Type

All elements within Processor ES1p have the same element type name, SHELL, thus:

```
RESET ELEMENT_TYPE = SHELL
```

should be entered before using the DEFINE ELEMENTS command. To select the 4-node, 9-node, or 16-node LAG shell element, use the /P qualifier in the DEFINE ELEMENTS command, i.e.,

```
DEFINE ELEMENTS /P = p
```

where *p* is the polynomial order and may be set to 1 (for the 4-LAG element), 2 (for the 9-LAG element) or 3 (for the 16-LAG element).

7.6.2.2 RESET Command for Element-Specific Research Parameters

None.

7.6.2.3 RESET Commands for Drilling Stiffness and Angle Tolerance

The default *scale* parameter used to compute artificial drilling stiffness is 5, which corresponds to a scale factor of 10-5 (see Figure 7.6-4). The value of scale can be changed via the RESET DRILL_STIFF command.

The default angle tolerance for requiring artificial drilling stiffness is 1 degree. Drilling stiffness flags are turned on at any node for which the normals of all attached shell elements make an angle less than this tolerance with the average element normal. The default tolerance can be changed via the RESET DRILL_TOL command.

Both of the above parameters also appear in the AUTO_DRILL solution procedure argument, and the angle tolerance parameter appears in the AUTO_TRIAD and AUTO_DOF_SUP solution procedure arguments.

7.6.3 Element Input/Output Datasets

General input and output dataset specifications are inherited from the generic element processor (see Section 7.2). There are no special-purpose datasets or attributes at this time.

7.6.3.1 Auxilliary Storage Dataset

None.

7.6.3.2 Other Special-Purpose Datasets/Attributes

None.

7.6.4 Element Implementation Status and Limitations

A summary of the current implementation status of the LAG shell elements within processor ES1p is given in Table 7.6-4. All functions except for the load stiffness matrix and element-dependent error estimates are implemented for all element types. Neither of these functions is essential. Generic element error estimates are adequate for adaptive refinement, and the load stiffness matrix is important only for some buckling eigenproblems involving live loads (e.g., hydrostatically loaded cylindrical shells).

Table 7.6-4 Processor ES1p Element Implementation Status

Functions	p=3 (4-node) LAG Element Status	p=2 (9-node) LAG Element Status	p=3 (16-node) LAG Element Status
Auto DOF Supp.	Yes	Yes	Yes
Body Forces	Yes	Yes	Yes
Consistent Mass	Yes	Yes	Yes
Diagonal Mass	Yes	Yes	Yes
Drilling Stiffness	Yes	Yes	Yes
Error Estimates/Elt-dep.	No	No	No
Error Estimates/Generic	Yes	Yes	Yes
Geometric Nonlinearity	Yes	Yes	Yes
Geometric Stiffness	Yes	Yes	Yes
Internal Forces	Yes	Yes	Yes
Load Stiffness	No	No	No
Material Nonlinearity	Yes	Yes	Yes
Pressure Forces	Yes	Yes	Yes
Strains	Yes	Yes	Yes
Stresses	Yes	Yes	Yes
Stress Extrapolation	Yes	Yes	Yes
Stress Transformation	Yes	Yes	Yes
Surface Forces	Yes	Yes	Yes

Higher-order LAG shell elements (beyond p=3) are implemented internally, and can be activated by recreating processor ES1p with a modified include block (contact the COMET-AR development team for details).

7.6.5 Element Error Messages

A summary of the most important or common error messages associated specifically with processor ES1p are described in Table 7.6-5.

Table 7.6-5 Summary of Element Processor ES1p Error Messages

Error #	Error Message	Probable Cause(s)	Recommended User Response
1	Invalid ES1p element type	The user has selected an invalid element type (via the RESET ELEMENT_TYPE command) when defining element connectivity or loads.	Change the element type to SHELL, as that is currently the only valid element type in processor ES1p.
2	p level exceeds current limit	The user has selected the element polynomial order (via the /P qualifier in the DEFINE ELEMENTS command) that is beyond a hardcoded limit in processor ES1p.	Reduce the polynomial order to an acceptable value, less than or equal to 5 on most installations, or ask the COMET-AR development team to increase the hardcoded limit.
3	ES0**** not implemented	The element developer has not implemented this element function.	Try to work around the unimplemented function; or ask the element developer to implement it ASAP.
4	Zero determinant of Jacobian	The element nodes probably do not define a proper quadrilateral. Either the nodal coordinates are not as intended by the user, or the definition of element nodal connectivity via the DEFINE ELEMENTS command is incorrect.	Check nodal coordinates and element connectivity. (Error is probably *not* due to the degeneration of a quadrilateral into a triangle; that is a permissible modeling technique with this element processor.)

7.6.6 Element Selection and Usage Guidelines

The following element selection and usage guidelines are based on experience to-date with COMET-AR Processor ES1p.

7.6.6.1 Element Type Selection

Of the three standard element types within Processor ES1p (4-LAG, 9-LAG, and 16-LAG), the 16-LAG element is the most robust, followed by the 9-LAG element. The 4-LAG is not recommended unless 4-node elements are absolutely necessary (e.g., from a large pre-existing model). Since the 9-LAG and 16-LAG are fully integrated isoparametric elements, they tend to be stiff for thin, curved shells (compared to the corresponding ANS elements in Processor ES7p). In contrast, the 4-LAG element, which would be unacceptably stiff with full integration (i.e., it would lock), is treated with selective/reduced integration. This makes the element more flexible, but at the expense of engendering potential spurious kinematic modes. Any model run with the 4-LAG (p=1) element should be examined first for spurious modes via a preliminary eigenvalue (e.g., vibration) analysis.

7.6.6.2 Problem Class Recommendations

The 9-LAG and 16-LAG elements in Processor ES1p are usable for general-purpose linear/ nonlinear analysis, but not optimal. For improved accuracy and reliability, use the ANS shell elements in Processor ES7p.

7.6.6.3 Distortion Sensitivity

The LAG quadrilateral shell elements do have some sensitivity to mesh distortion (i.e., element corner point angles that are far from 90 degrees), but they are less sensitive than a number of other shell elements (e.g., those in processor ES7p). However, the overall accuracy of the LAG elements may be significantly less than these other, more distortion-sensitive elements for the same mesh.

7.6.6.4 Automatic Drilling Stabilization

Since the LAG shell elements do not have intrinsic drilling rotational stiffness, the user must select one of the automatic drilling DOF stabilization options available in COMET-AR solution procedures: either the AUTO_DRILL option (which will engender artificial drilling stiffness at the element level); or the AUTO_DOF_SUP option (which will suppress global rotational DOFs if the computational axes are closely aligned with the element normal). The AUTO_TRIAD option may also be selected in conjunction with the AUTO_DOF_SUP option, if the computational axes are not closely aligned with the element normals. At shell/shell, or shell/stiffener junctures, drilling stabilization is unnecessary.

7.6.6.5 Adaptive Analysis Guidelines

All of the LAG shell elements in Processor ES1p may be used in conjunction with adaptive mesh refinement (AR) with the following provisos:

1) The 4-LAG element may be sensitive to the multipoint interelement constraints generated by constraint-based (h_c) refinement. This is because the selective/reduced integration used on this element makes it effectively an incompatible (i.e., non-conforming) element.

2) The 9-LAG and 16-LAG elements seem to work well with either hc or ht refinement; however, they usually converge much more slowly than the corresponding ANS elements in processor ES7p.

7.6.7 References

[1] Stanley, G. M., *The Computational Structural Mechanics Testbed (COMET) Structural Element Processor ES1: Basic SRI and ANS Shell Elements*, NASA CR 4357, 1991.

[2] Stanley, G. M., "Continuum-Based Shell Elements," Ph.D. Thesis, Stanford University, Stanford, CA, 1985.

[3] Stanley, G. M. and Nour-Omid, S., *The Computational Structural Mechanics Testbed (COMET) Generic Structural-Element Processor Manual*, NASA CR 181728, 1990.

7.7 Processor ES7p (Variable-p ANS Quadrilateral Shell Elements)

7.7.1 Element Description

Processor ES7p contains a family of variable-polynomial (p)-order, assumed natural-coordinate strain (ANS) quadrilateral shell elements, ranging from a 4-node element (p=1) to a 16-node element (p=3). The formulation is based on an extension of the ANS elements described in [1] and [2], which features a C^0 (shear-deformable) shell theory [3] with directionally selective approximations for each natural-coordinate strain component. The result is a set of rapidly convergent elements that are insensitive to element thickness-to-length aspect ratios, and free of spurious kinematic modes. These shell elements may also be viewed as high-performance extensions of displacement-based isoparametric (Lagrange) shell elements (e.g., those in processor ES1p), as they use an implicit form of directionally reduced numerical integration on the basic Lagrange elements to achieve increased accuracy, prevent locking and avoid mechanisms. On the other hand, the lower-order elements (p=1 and p=2) tend to be somewhat more sensitive to mesh distortion than the corresponding basic Lagrange elements

7.7.1.1 Summary of Element Types

Currently implemented element types available within processor ES7p are summarized in Table 7.7-1.

Table 7.7-1 Summary of Processor ES7p Element Types

Element Type Name	Description	Status
SHELL (p=1)	4-node ANS quadrilateral shell element; bilinear geometry and displacement field; constant/linear strain field.	Implemented
SHELL (p=2)	9-node ANS quadrilateral shell element; biquadratic geometry and displacement field; linear/quadratic strain field.	Implemented
SHELL (p=3)	16-node ANS quadrilateral shell element; bicubic geometry and displacement field; quadratic/cubic strain field.	Implemented

7.7.1.2 Element Geometry and Node Numbering

The three ANS shell element types listed in Table 7.7-1 are illustrated in Figures 7.7-1 to 7.7-3. Element nodes are shown as solid circles with bold node numbers and integration (stress-storage) points are shown as X's with plain number subscripts. Element boundary (line) numbers and node numbering conventions within boundaries (for line load application) are shown in part **b** of each figure.

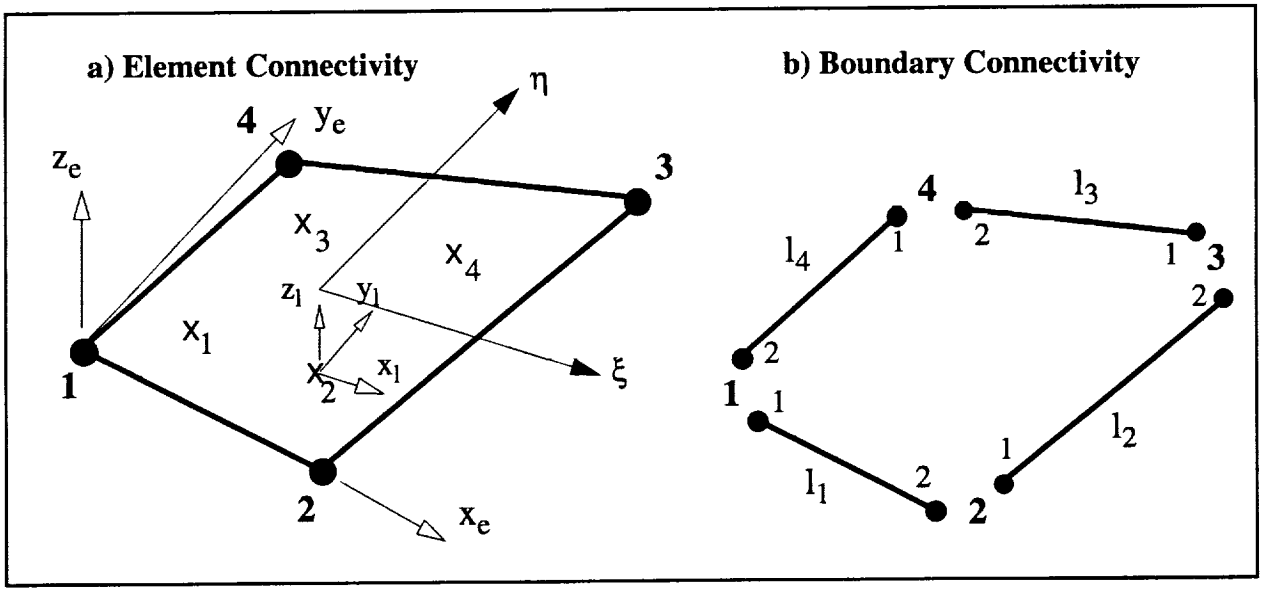


Figure 7.7-1 4-ANS (p=1) Element Geometry and Node Numbers

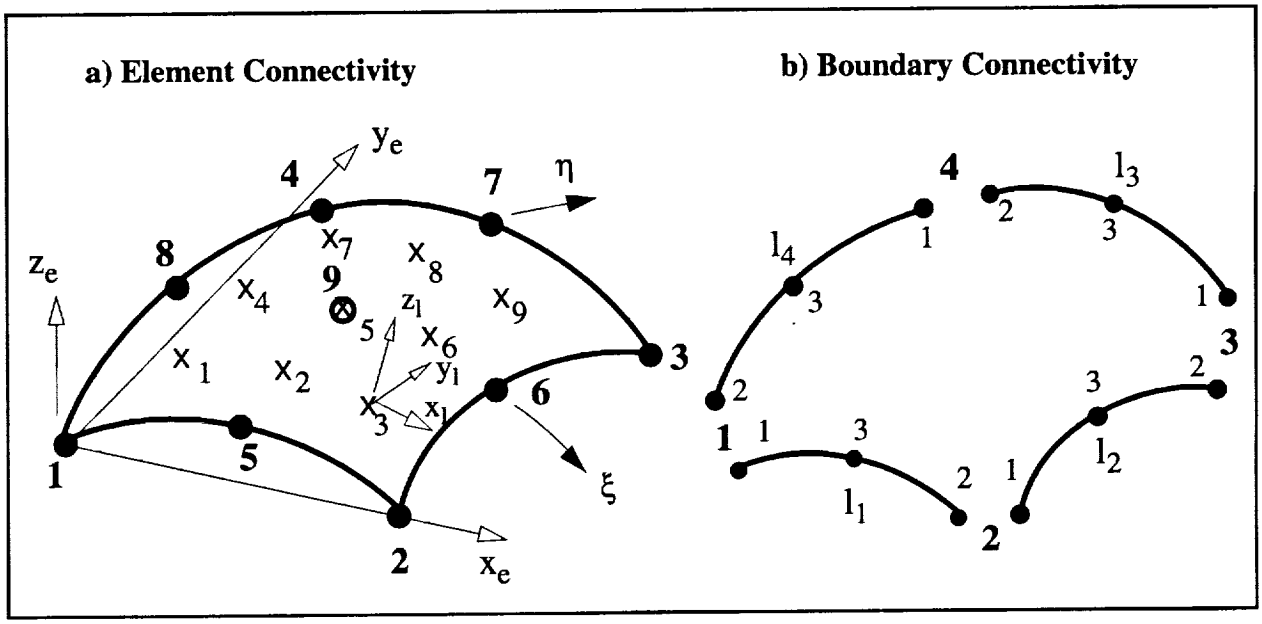


Figure 7.7-2 9-ANS (p=2) Element Geometry and Node Numbers

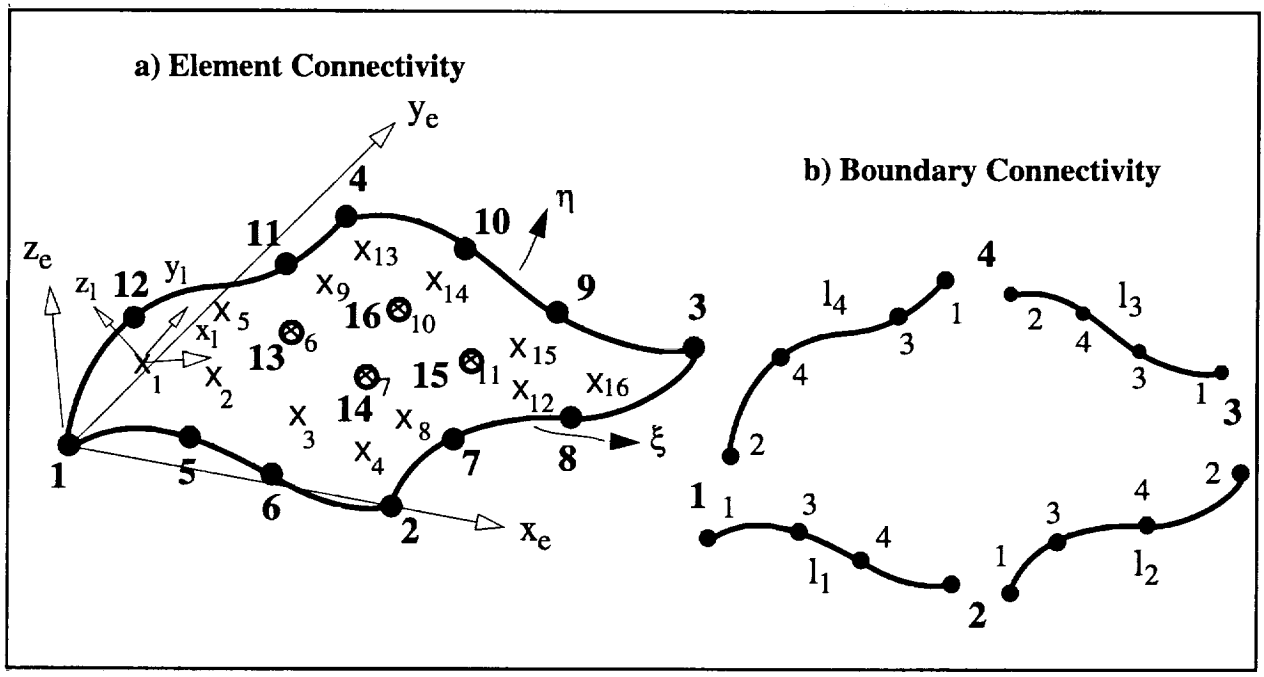


Figure 7.7-3 16-ANS (p=3) Element Geometry and Node Numbers

In Figures 7.7-1 to 7.7-3, the orthogonal x_e, y_e, z_e axes form the element Cartesian (or corotational) coordinate system. The orthogonal x_l, y_l, z_l axes form the element local stress coordinate system, which can vary from integration point to integration point; and the non-orthogonal/curvilinear ξ, η, ζ axes from the element natural-coordinate system. The x_e axis initially connects nodes 1 and 2, and the z_e axis is perpendicular to the 1-2-3 plane; however, this coordinate system is slightly modified by the generic element processor to achieve a less biased system for corotational nonlinear analysis (see Reference [4]). The x_l axis is always tangent to the local ξ curve, the z_l axis is always normal to the ξ–η tangent plane, and the y_l axis completes an orthogonal triad.

7.7.1.3 Nodal Freedoms (DOFs) and BCs

All of the quadrilateral shell elements in Processor ES7p have 3 translational displacement DOFs and 3 rotational displacement DOFs at each element node (see Figure 7.7-4). The drilling rotational DOF (i.e., the rotation about the local element surface-normal vector) does not have any intrinsic stiffness, and one of two drilling stabilization options must be employed with this element: i) artificial drilling stiffness (which may be triggered via the AUTO_DRILL option at the solution procedure level), or ii) automatic drilling DOF suppression via the AUTO_DOF_SUP option in conjunction with the AUTO_TRIAD option for models (see Section 2.10, *Automatic DOF Suppression and Drilling Stabilization*).

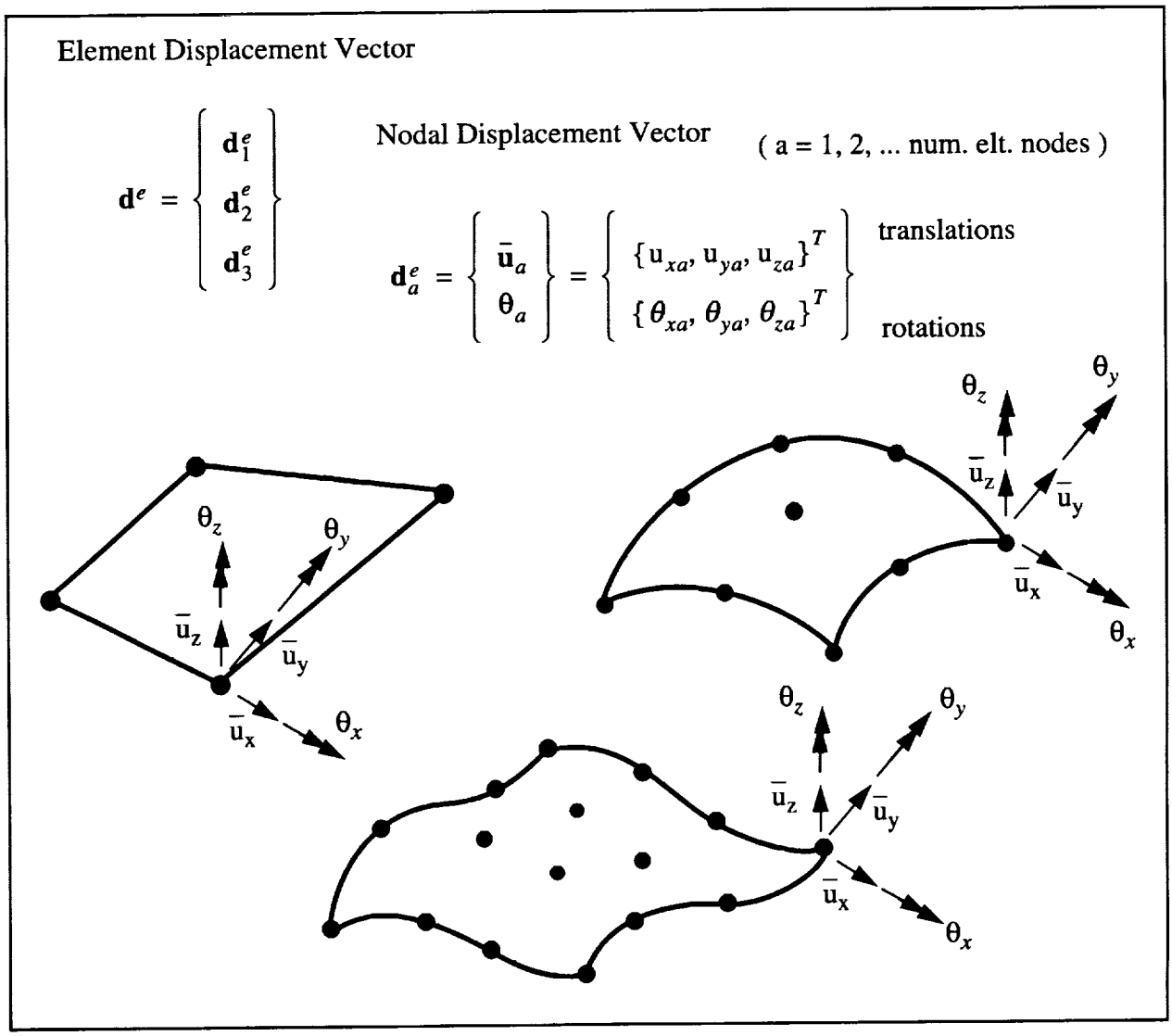


Figure 7.7-4 Displacement DOFs for ANS Shell Elements

7.7.1.4 Displacement Representation

The approximation of the displacement field within ANS shell elements, assumed independently of the strain field, is shown in Table 7.7-2

Table 7.7-2 Processor ES7p Element Displacement Approximations

Component	Polynomial Variation		
	p = 1 (4-node)	p = 2 (9-node)	p = 3 (16-node)
$\bar{u}(\xi,\eta)$	Lin(ξ)*Lin(η)	Quad(ξ)*Quad(η)	Cubic(ξ)*Cubic(η)
$\theta(\xi,\eta)$	Lin(ξ)*Lin(η)	Quad(ξ)*Quad(η)	Cubic(ξ)*Cubic(η)

7.7.1.5 Strain Representation

The ANS shell elements in Processor ES7p generate 8 resultant strain components, which are stored at each of the element integration points. The 8-strain resultants are arranged as follows:

$$\varepsilon = \begin{bmatrix} \bar{\varepsilon} \\ \kappa \\ \gamma \end{bmatrix} = \begin{bmatrix} \text{Membrane_Strains} \\ \text{Bending_Strains} \\ \text{Transverse-Shear_Strains} \end{bmatrix}$$

where

$$\bar{\varepsilon} = \begin{bmatrix} \bar{\varepsilon}_x \\ \bar{\varepsilon}_y \\ \bar{\varepsilon}_{xy} \end{bmatrix} \qquad \kappa = \begin{bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix} \qquad \gamma = \begin{bmatrix} \gamma_x \\ \gamma_y \end{bmatrix}$$

where the subscripts x and y denote the x_l and y_l components at an integration point (see Figures 7.7-1 to 7.7-3). The variation of the natural-coordinate components of these strains within the element domain is assumed independently of the displacement field, as summarized in Table 7.7-3.

Table 7.7-3 Processor ES7p Element Strain Approximations

Component	Polynomial Variation		
	p = 1 (4-node)	**p = 2 (9-node)**	**p = 3 (16-node)**
ε_ξ, k_ξ, γ_ξ	Lin(η)	Lin(ξ)*Quad(η)	Quad(ξ)*Cubic(η)
ε_η, κ_η, γ_η	Lin(ξ)	Quad(ξ)*Lin(η)	Cubic(ξ)*Quad(η)
$\bar{\varepsilon}_{\xi\eta}$, $\bar{\kappa}_{\xi\eta}$	Constant	Lin(ξ)*Lin(η)	Quad(ξ)*Quad(η)

7.7.1.6 Stress Representation

Stress resultants conjugate to the above strain resultants are computed via the Generic Constitutive Processor (GCP), and are arranged as follows:

$$\sigma = \begin{bmatrix} \mathbf{N} \\ \mathbf{M} \\ \mathbf{Q} \end{bmatrix} = \begin{bmatrix} \text{Membrane_Stresses} \\ \text{Bending_Stresses} \\ \text{Transverse-Shear_Stresses} \end{bmatrix}$$

where

$$\mathbf{N} = \begin{bmatrix} N_x \\ N_y \\ N_{xy} \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} M_x \\ M_y \\ M_{xy} \end{bmatrix} \qquad \mathbf{Q} = \begin{bmatrix} Q_x \\ Q_y \end{bmatrix}$$

Like the strains, the stress resultants are also computed and stored at the element integration points, and have the same polynomial variations (for linear constitutive models).

7.7.1.7 Drilling Rotational Stiffness

Since the present shell element formulation has no intrinsic drilling (normal rotational) stiffness, an artificial drilling stiffness option is provided. This option is triggered by the AUTO_DRILL solution procedure argument, and works as shown in Figure 7.7-5.

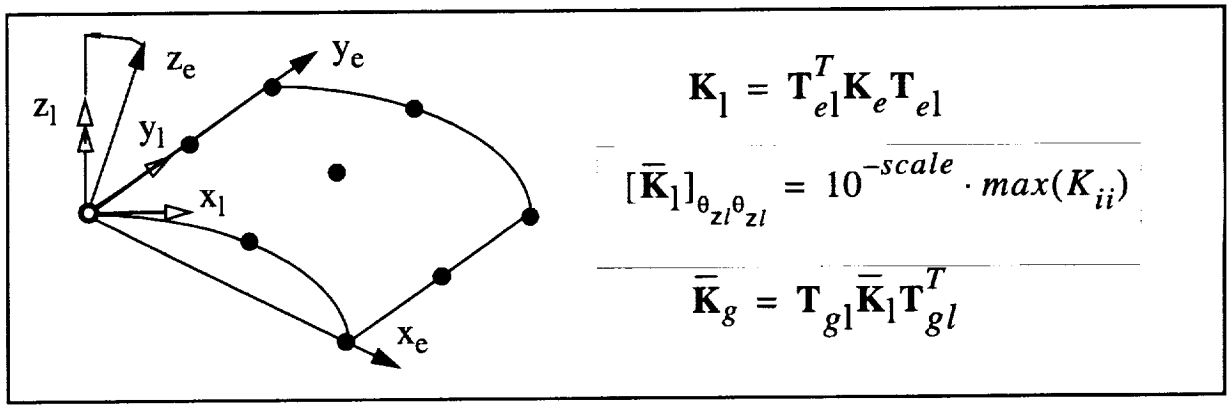


Figure 7.7-5 Implementation of Artificial Drilling Stiffness in Processor ES7p

The element material stiffness matrix is first computed in the element corotational frame (x_e, y_e, z_e) and then rotated into an independent local frame (l) at each node such that the z_l axis is parallel to the element normal (or drilling) axis. The diagonal drilling rotational stiffness components are then set equal to a small fraction of the maximum element diagonal stiffness component. Finally, the element matrix is rotated back to the element corotational frame before depositing in the database for assembly. The fractional coefficient multiplying the maximum diagonal stiffness component involves a negative power of 10. That exponent, referred to as *scale*, corresponds to the *scale* parameter in the AUTO_DRILL solution procedure argument (and also in the element processor's RESET DRILL_STIFF command). The default coefficient is 10-5 (*scale*=5).

7.7.1.8 Element Nonlinearity

Element geometrical nonlinearity is accounted for by an Updated Lagrangian treatment of the element force vector and stiffness matrix, and by a moderate-rotation nonlinear strain measure based on the midpoint strain tensor. Additionally, the ANS shell elements may be employed with the generic element processors (ES) built-in corotational capability to enable arbitrarily large rotations. For material nonlinearity, Processor ES7p is fully compatible with the generic constitutive processor, and all specific shell constitutive models implemented therein.

7.7.2 Element Command Specifications

General command syntax and options are all inherited from the generic element processor (see Section 7.2). Special command options for Processor ES7p are described in the following subsections.

7.7.2.1 RESET Command for Element Type

All elements within Processor ES7p have the same element type name, SHELL, thus:

```
RESET ELEMENT_TYPE = SHELL
```

should be entered before using the DEFINE ELEMENTS command. To select the 4-node, 9-node, or 16-node ANS shell element, use the /P qualifier in the DEFINE ELEMENTS command, i.e.,

```
DEFINE ELEMENTS /P = p
```

where p is the polynomial order and may be set to 1 (for the 4-ANS element), 2 (for the 9-ANS element) or 3 (for the 16-ANS element).

7.7.2.2 RESET Command for Element-Specific Research Parameters

None.

7.7.2.3 RESET Commands for Drilling Stiffness and Angle Tolerance

The default *scale* parameter used to compute artificial drilling stiffness is 5, which corresponds to a scale factor of 10-5 (see Figure 7.7-4). The value of *scale* can be changed via the RESET DRILL_STIFF command.

The default angle tolerance for requiring artificial drilling stiffness is 1 degree. Drilling stiffness flags are turned on at any node for which the normals of all attached shell elements make an angle less than this tolerance with the average element normal. The default tolerance can be changed via the RESET DRILL_TOL command.

Both of the above parameters also appear in the AUTO_DRILL solution procedure argument, and the angle tolerance parameter appears in the AUTO_TRIAD and AUTO_DOF_SUP solution procedure arguments.

7.7.3 Element Input/Output Datasets

General input and output dataset specifications are inherited from the generic element processor (see Section 7.2). There are no special-purpose datasets or data attributes at this time.

7.7.3.1 Auxilliary Storage Dataset

None.

7.7.3.2 Other Special-Purpose Datasets/Attributes

None.

7.7.4 Element Implementation Status and Limitations

A summary of the current implementation status of the ANS shell elements within processor ES7p is given in Table 7.7-4. All functions except for the load stiffness matrix and element-dependent error estimates are implemented for all element types. Neither of these functions is essential. Generic element error estimates are adequate for adaptive refinement, and the load stiffness matrix is important only for some buckling eigenproblems involving live loads (e.g., hydrostatically loaded cylindrical shells).

Table 7.7-4 Processor ES7p Element Implementation Status

Functions	p=3 (4-node) ANS Element Status	p=2 (9-node) ANS Element Status	p=3 (16-node) ANS Element Status
Auto DOF Supp.	Yes	Yes	Yes
Body Forces	Yes	Yes	Yes
Consistent Mass	Yes	Yes	Yes
Diagonal Mass	Yes	Yes	Yes
Drilling Stiffness	Yes	Yes	Yes
Error Estimates/Elt-dep.	No	No	No
Error Estimates/Generic	Yes	Yes	Yes
Geometric Nonlinearity	Yes	Yes	Yes
Geometric Stiffness	Yes	Yes	Yes
Internal Forces	Yes	Yes	Yes
Load Stiffness	No	No	No

Table 7.7-4 Processor ES7p Element Implementation Status (Continued)

Functions	p=3 (4-node) ANS Element Status	p=2 (9-node) ANS Element Status	p=3 (16-node) ANS Element Status
Material Nonlinearity	Yes	Yes	Yes
Pressure Forces	Yes	Yes	Yes
Strains	Yes	Yes	Yes
Stresses	Yes	Yes	Yes
Stress Extrapolation	Yes	Yes	Yes
Stress Transformation	Yes	Yes	Yes
Surface Forces	Yes	Yes	Yes

Higher-order ANS shell elements (beyond p=3) are implemented internally and can be activated by recreating processor ES7p with a modified include block (contact COMET-AR development team for details).

7.7.5 Element Error Messages

A summary of the most important or common error messages associated specifically with processor ES7p are described in Table 7.7-5.

Table 7.7-5 Summary of Element Processor ES7p Error Messages

Error #	Error Message	Probable Cause(s)	Recommended User Response
1	Invalid ES1p element type	The user has selected an invalid element type (via the RESET ELEMENT_TYPE command) when defining element connectivity or loads.	Change the element type to SHELL, as that is currently the only valid element type in processor ES7p.
2	p level exceeds current limit	The user has selected the element polynomial order (via the /P qualifier in the DEFINE ELEMENTS command) that is beyond a hardcoded limit in processor ES1p.	Reduce the polynomial order to an acceptable value, which is less than or equal to 5 on most installations, or as the COMET-AR development team to increase the hardcoded limit.
3	ES0 **** not implemented	The element developer has not implemented this particular element function.	Try to work around the unimplemented function; or ask the element developer to implement it ASAP.

Table 7.7-5 Summary of Element Processor ES7p Error Messages (Continued)

Error #	Error Message	Probable Cause(s)	Recommended User Response
4	Zero determinant of Jacobian	The element nodes probably do not define a proper quadrilateral. Either the nodal coordinates are not as intended by the user, or the definition of element nodal connectivity via the DEFINE ELEMENTS command is incorrect.	Check nodal coordinates and element connectivity. (Error probably *not* due to the degeneration of a quadrilateral into a triangle, a permissible modeling technique with this element processor.)

7.7.6 Element Selection and Usage Guidelines

The following element selection and usage guidelines are based on experience to-date with COMET-AR Processor ES7p.

7.7.6.1 Element Type Selection

Of the three standard element types within Processor ES7p (4-ANS, 9-ANS, and 16-ANS), the 16-ANS element is the most robust, followed by the 9-ANS element. The 4-ANS is not recommended unless 4-node elements are absolutely necessary (e.g., from a large pre-existing model). All of these ANS shell elements are significantly more accurate than their counterpart Lagrange elements (in processor ES1p) unless there is significant mesh distortion (see Distortion Sensitivity below).

7.7.6.2 Problem Class Recommendations

The ANS elements in Processor ES7p are recommended for general-purpose linear/nonlinear analysis. For curved structures, the higher-order (p=2, 3) elements are strongly recommended.

7.7.6.3 Distortion Sensitivity

The 4-ANS and 9-ANS (p=1 and p=2) elements are more distortion-sensitive than their Lagrange counterparts (e.g., in processor ES1p); however, the 16-ANS (p=3) element is much less distortion sensitive than lower-order elements, and similar in this regard to the 16-LAG element.

7.7.6.4 Automatic Drilling Stabilization

Since the ANS shell elements do not have intrinsic drilling rotational stiffness, the user must select one of the automatic drilling DOF stabilization options (see Section 2.10) available in COMET-AR solution procedures: the AUTO_DRILL option (which will engender artificial drilling stiffness at the element level); or the AUTO_DOF_SUP option (which will suppress global rotational DOFs if the computational axes are closely aligned with the element normal). The AUTO_TRIAD option may also be selected in conjunction with the AUTO_DOF_SUP option, if the computational axes

are not closely aligned with the element normals. At shell/shell, or shell/stiffener junctures, drilling stabilization is unnecessary.

7.7.6.5 Adaptive Analysis Guidelines

All of the ANS shell elements in Processor ES7p may be used in conjunction with adaptive mesh refinement (AR) with the following provisos:

1) The 4-ANS (p=1) and 9-ANS (p=3) elements can be distortion-sensitive when used with transition-based (h_t) refinement; the 9-ANS is recommended over the 4-ANS.

2) The 4-ANS and 9-ANS elements are also sensitive to the multipoint constraints generated by constraint-based (h_c) refinement; again, the 9-ANS is recommended over the 4-ANS.

3) The 16-ANS is recommended with either h_c or h_t refinement; however, its storage requirements can be considerably higher than lower-order elements, especially when using a direct equation solvers. Iterative solvers are therefore recommended.

7.7.7 References

[1] Park, K. C. and Stanley, G. M., "A Curved C0 Shell Element Based on Assumed Natural Coordinate Strains," *Journal of Applied Mechanics*, Vol. 108, pp. 278-290, 1986.

[2] Stanley, G. M., Park, K. C., and Cabiness, H., *The Computational Structural Mechanics Testbed (COMET) Structural Element Processor ES7: Revised ANS Shell Elements*, NASA CR 4360, 1991.

[3] Stanley, G. M., "Continuum-Based Shell Elements," Ph.D. Thesis, Stanford University, Stanford, CA, 1985.

[4] Stanley, G. M. and Nour-Omid, S., *The Computational Structural Mechanics Testbed (COMET) Generic Structural-Element Processor Manual*, NASA CR 181728, 1990.

7.8 Processor ES36 (MIN3/6 Triangular Shell Elements)

7.8.1 Element Description

Processor ES36 contains a 3-node triangular shell element, called MIN3, based on Mindlin (moderately thick) plate theory, with an anisoparametric finite element displacement field to maintain element performance in both thin and thick shell limits. A curved shallow shell version of the element (MIN6) employing 6 nodes for geometric description is partially implemented. The formulation of both MIN3 and MIN6 elements are described in detail in Reference [1]. Processor ES36 was developed by Alex Tessler and Majdi Baddourah of NASA Langley Research Center, with assistance from Gary Stanley of Lockheed Palo Alto Research Laboratory.

7.8.1.1 Summary of Element Types

Element types currently implemented or under development that are available within processor ES36 are summarized in Table 7.8-1.

Table 7.8-1 Summary of Processor ES36 Element Types

Element Type Name	Description	Status
MIN3	3-node triangular plate element; can be used as a flat facet element approximation to model shell structures	Implemented
MIN6	6-node triangular shallow-shell element; only 3 active nodes, the other 3 nodes used exclusively for geometrical description	Under Development

7.8.1.2 Element Geometry and Node Numbering

The MIN3 element is illustrated in Figure 7.8-1 and the MIN6 element in Figure 7.8-2. Both elements have one centroidal integration point (for stress storage). The boundary node numbering conventions are the same for both MIN3 and MIN6, as the midside nodes appearing in the definition of the MIN6 nodal connectivity are used only for geometric purposes. No loads, boundary conditions, or displacements are present there. Element integration (i.e., stress/strain storage) points are indicated by an X designating the location, and a number in parentheses denoting the integration point numbering convention.

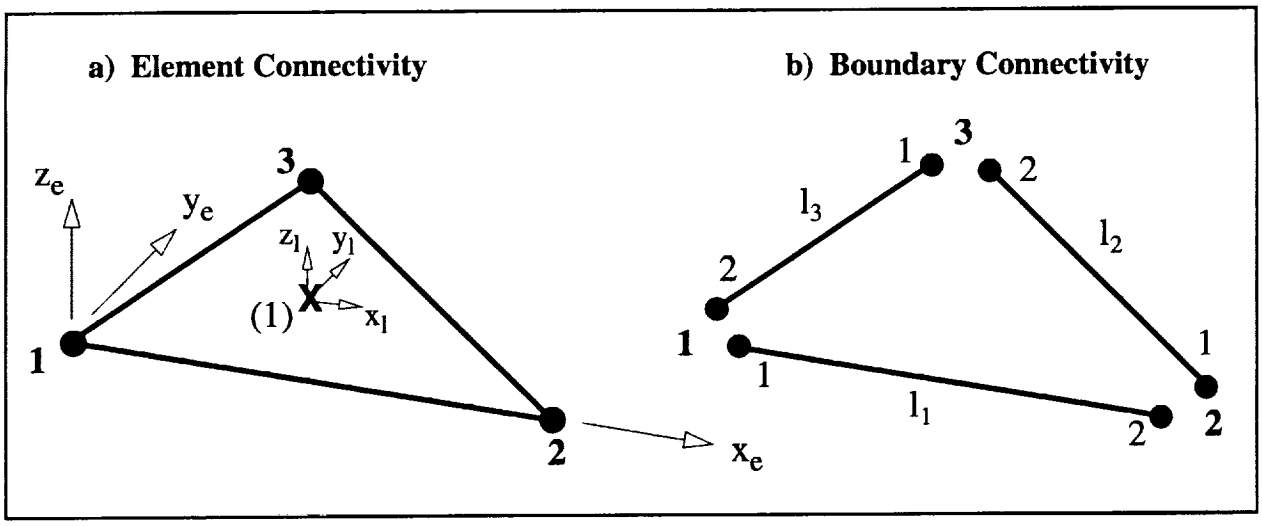


Figure 7.8-1 MIN3 Element Geometry and Node Numbers

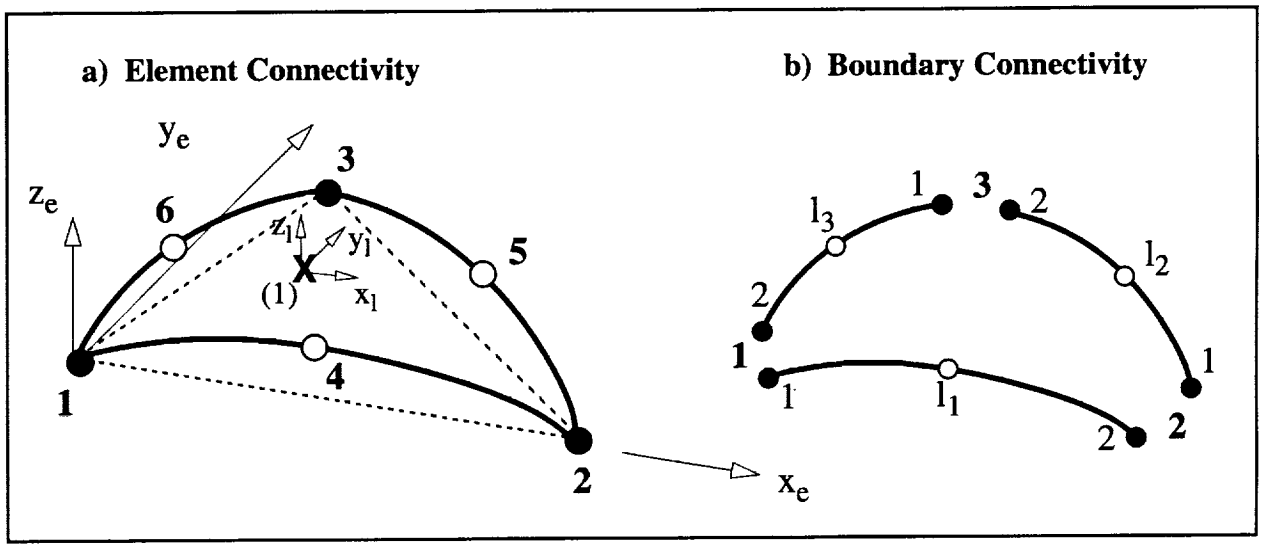


Figure 7.8-2 MIN3 Element Geometry and Node Numbers

7.8.1.3 Nodal Freedoms (DOFs) and BCs

Both the MIN3 and MIN6 shell elements have 3 translational displacement DOFs and 3 rotational displacement DOFs at each of the 3 element corner nodes (see Figure 7.8-3). The drilling rotational DOF (i.e., the rotation about the local element surface-normal vector) is provided with artificial stiffness so that the user does not have to be concerned about suppressing it.

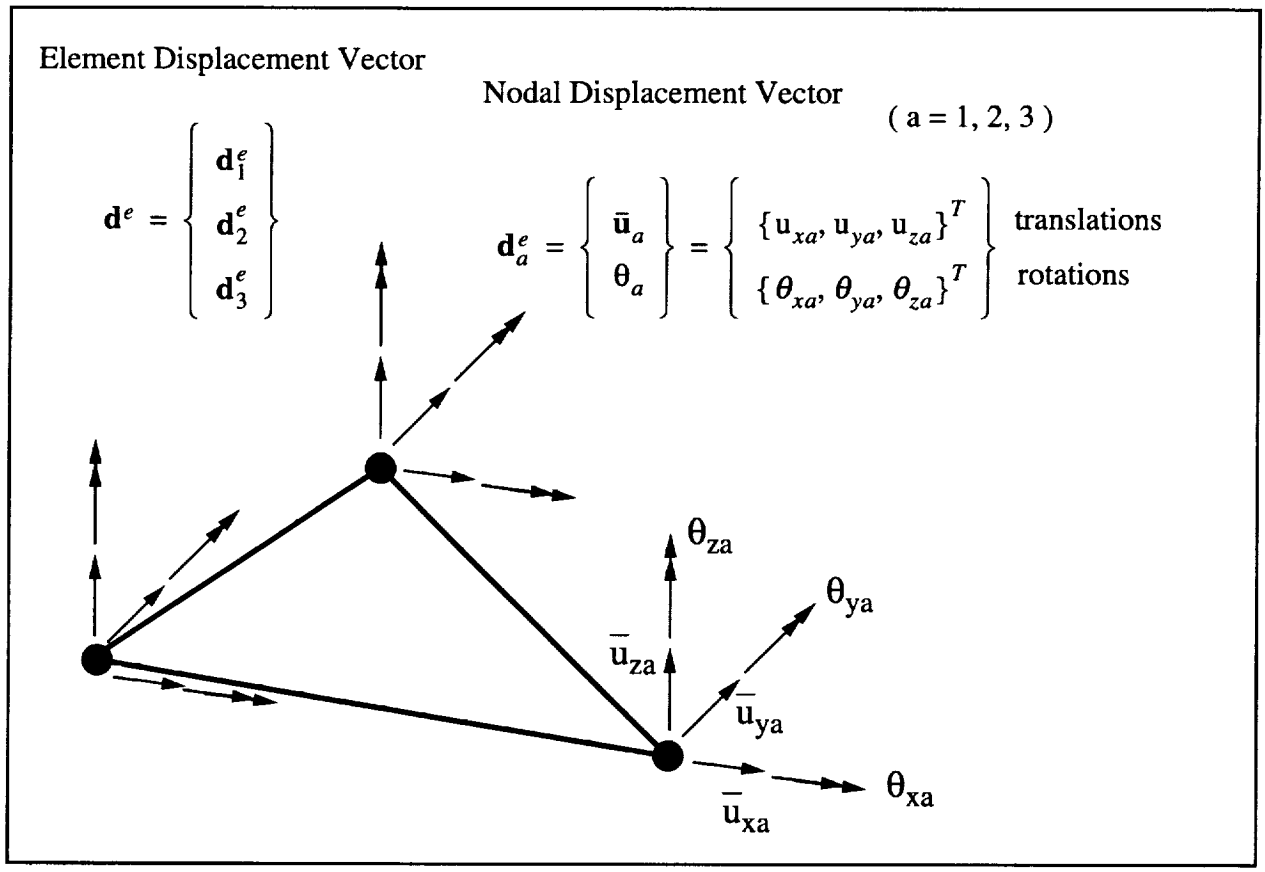


Figure 7.8-3 Displacement DOFs for MIN3 and MIN6 Elements

7.8.1.4 Displacement Representation

The approximation of the displacement field within the element is summarized in Table 7.8-2.

Table 7.8-2 MIN3 Element Displacement Approximation

Component	Polynomial Variation in x_e, y_e Plane
u_{xe}, u_{ye}, θ_{xe}, θ_{ye}	Linear
u_{ze}	Quadratic
θ_{ze}	Irrelevant

7.8.1.5 Strain Representation

The MIN3 and MIN6 shell elements each generate 8 resultant strain components, which are stored at the element centroid. The 8-strain resultants are arranged as follows:

$$\varepsilon = \begin{bmatrix} \bar{\varepsilon} \\ \kappa \\ \gamma \end{bmatrix} = \begin{bmatrix} \text{Membrane_Strains} \\ \text{Bending_Strains} \\ \text{Transverse-Shear_Strains} \end{bmatrix}$$

where

$$\bar{\varepsilon} = \begin{bmatrix} \bar{\varepsilon}_x \\ \bar{\varepsilon}_y \\ \bar{\varepsilon}_{xy} \end{bmatrix} \qquad \kappa = \begin{bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix} \qquad \gamma = \begin{bmatrix} \gamma_x \\ \gamma_y \end{bmatrix}$$

and are expressed in the local centroidal x_l, y_l, z_l system, which coincides with the element corotational frame: x_e, y_e, z_e (see Figure 7.8-1). The variation of these strain components within the element domain is summarized in Table 7.8-3.

Table 7.8-3 MIN3 Element Displacement Approximation

Component	Polynomial Variation in x_e, y_e Plane
$\bar{\varepsilon}$	Linear
κ	Linear
γ	Linear

7.8.1.6 Stress Representation

Stress resultants conjugate to the above strain resultants are computed via the generic constitutive processor, and are arranged as follows:

$$\sigma = \begin{bmatrix} \mathbf{N} \\ \mathbf{M} \\ \mathbf{Q} \end{bmatrix} = \begin{bmatrix} \text{Membrane_Stresses} \\ \text{Bending_Stresses} \\ \text{Transverse-Shear_Stresses} \end{bmatrix}$$

where

$$\mathbf{N} = \begin{bmatrix} N_x \\ N_y \\ N_{xy} \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} M_x \\ M_y \\ M_{xy} \end{bmatrix} \qquad \mathbf{Q} = \begin{bmatrix} Q_x \\ Q_y \end{bmatrix}$$

Like the strains, the stress resultants are also saved at the element centroid, although the internal variation within an element is linear (for linear constitutive models).

7.8.1.7 Drilling Rotational Stiffness

Since there is no intrinsic normal rotational (i.e., drilling) stiffness associated with the present shell element formulation, artificial drilling stiffness is added to stabilize drilling DOFs. At each element node a tiny fraction (10^{-6}) of the smallest physical component on the diagonal of the element stiffness matrix is added to the diagonal term corresponding to the drilling DOF. This addition to the element stiffness matrix is hardwired internally, so that the user cannot change its magnitude (or angle tolerance) via COMET-AR's AUTO_DRILL option.

7.8.1.8 Element Nonlinearity

Element geometrical nonlinearity is presently not accounted; large rotation effects must therefore be relegated to the corotational option within the COMET-AR generic element processor. Material nonlinearity is accommodated via the COMET-AR generic constitutive processor.

7.8.2 Element Command Specifications

General command syntax and options are all inherited from the generic element processor (see Section 7.2). Special command options peculiar to Processor ES36 are described in the following subsections.

7.8.2.1 RESET Element Research Parameters

None.

7.8.2.2 RESET Drilling Stiffness and Angle Tolerances

Irrelevant for the elements in this processor (artificial drilling stiffness is hardwired).

7.8.3 Element Input/Output Datasets

General input and output dataset specifications are inherited from the generic element processor (see Section 7.2). Any special-purpose datasets or data attributes are discussed below.

7.8.3.1 Auxiliary Storage Dataset

Processor ES36 employs an auxiliary storage dataset, called:

ES36_MIN*nen*.AUX_STORAGE...*mesh*

where *nen* (the number of element nodes) is 3 for the MIN3 element, and 6 for the MIN6 element. This dataset, which is of class EAT, contains the initial constitutive matrix for each element in the

model, and is used later by the element to construct material-dependent relaxation parameters that improve the elements' performance in the thin and thick shell limits.

7.8.3.2 Other Special-Purpose Datasets/Attributes

None.

7.8.4 Element Implementation Status and Limitations

A summary of the current implementation status of the MIN3 and MIN6 elements within processor ES36 is given in Table 7.8-4. Only linear static analysis capabilities are currently available.

Table 7.8-4 Processor ES36 Element Implementation Status

Functions	MIN3 Status	MIN6 Status
Auto DOF Suppression	Yes	No
Body Forces	No	No
Consistent Mass	No	No
Diagonal Mass	No	No
Drilling Stiffness	Yes	No
Error Estimates/Elt-dep.	No	No
Error Estimates/Generic	Yes	No
Geometric Nonlinearity	No	No
Geometric Stiffness	No	No
Internal Forces	No	No
Load Stiffness	No	No
Material Nonlinearity	Yes	No
Pressure Forces	Yes	No
Strains	Yes	No
Stresses	Yes	No
Stress Extrapolation	Yes	No
Stress Transformation	Yes	No
Surface Forces	No	No

7.8.5 Element Error Messages

There are currently no special-purpose error messages associated with either the MIN3 or MIN6 elements in processor ES36.

7.8.6 Element Selection and Usage Guidelines

The following element selection and usage guidelines are based on minimal experience with the MIN3 and MIN6 elements within COMET-AR.

7.8.6.1 Element Type Selection

Presently, only the MIN3 element is operational. This element may be used for flat or curved shell structures, but due to the faceted approximation, a fine initial mesh may be required to capture geometric accuracy for curved structures.

7.8.6.2 Problem Class Recommendations

The MIN3 element is recommended for general smooth shell analysis, including transverse-shear deformation effects; however, for junctured or stiffened shells the built-in artificial drilling stiffness may decrease solution accuracy.

7.8.6.3 Distortion Sensitivity

Not yet evaluated.

7.8.6.4 Automatic Drilling Stabilization

The built-in artificial drilling stiffness of the MIN3 element means that the AUTO_DOF_SUP, AUTO_DRILL and AUTO_TRIAD solution procedure options have no effect on this element type.

7.8.7 References

[1] Tessler, A., "A C0-Anisoparametric Three-Node Shallow Shell Element for General Shell Analysis," Army Materials Technology Laboratory Report, MTL TR 89-72, 1989.

8 Constitutive Processors

8.1 Overview

This chapter describes the constitutive processing capabilities available in COMET-AR. These capabilities are implemented in a processor named the Generic Constitutive Processor (GCP). Table 8.1-1 shows the contents of this chapter.

Table 8.1-1 Outline of Chapter 8: Constitutive Processors

Section	Description
8.1	Overview
8.2	Generic Constitutive Processor Description
8.3	Fabrication Definition
8.4	Material Property Definition
8.5	Analysis Control
8.6	Update Command

8.2 Generic Constitutive Processor Description

8.2.1 General Description

The Generic Constitutive Processor (GCP) is a set of software modules that perform all constitutive functions for COMET-AR. Two functions are served by the GCP: as a stand-alone processor for use in testing of new constitutive models; and as a Fortran callable constitutive library directly accessible by COMET-AR element processors. To enable this duality in function, the GCP performs constitutive calculations using input received through a common interface from either the Generic Element Processor (GEP) during a finite element analysis, or from the GCP processor shell when operating in stand-alone mode.

The GCP provides a flexible, easy to use framework for the testing and incorporation of new constitutive modeling capability into COMET-AR. From a method developers standpoint, the GCP allows automatic access to constitutive functions by all element developers using the GEP, and conversely allows constitutive models incorporated by material researchers to be available to all elements implemented within the Generic Element Processor (GEP) framework. These capabilities have been included through standard developer interfaces described in detail in the GCP Manual.

Processor GCP is normally invoked directly within the users model definition procedure when used in conjunction with COMET-AR.

8.2.2 Command Summary

The GCP commands fall into four categories: fabrication definition; material definition; stand-alone analysis; and historical data update. The fabrication definition commands define the geometry of the structure at an element integration point, including pointers to the material properties to be used in the analysis. The material definition commands allow for the input of the properties or parameters that are used by a specific constitutive model, e.g., mechanical material properties, failure parameters, etc. The analysis commands are used to control a pointwise constitutive analysis when the GCP is used as a stand-alone processor. The historical data update command updates the constitutive historical data upon completion of a nonlinear load step.

Each of the four command categories is performed by a separate command subprocessor within the GCP . The commands used to enter each of these subprocessors is given in Table 8.2-1.

Table 8.2-1 Processor GCP Command Summary

Command Name	Function
FABRICATION	Initiate the fabrication subprocessor for fabrication definition
MATERIAL	Initiate the material subprocessor for material property definition
ANALYSIS	Initiate the analysis subprocessor for pointwise constitutive analysis
UPDATE	Update constitutive historical databases for a load step

8.2.3 Database Input/Output

8.2.3.1 Input Datasets

The GCP does not use any input datasets.

8.2.3.2 Output Datasets

A summary of output datasets created by the GCP is given in Table 8.2-2.

Table 8.2-2 Output Datasets Created By The GCP

Dataset	Description
MATL.PNTR	Pointers to material property data
MATL.DATA	Material property definition data
FABRICATIONS	Fabrication definition data
CONTSTITUTIVE.STIFFNESS	Integrated constitutive stiffnesses for each fabrication
EltType.EHIST.*step*	Element based constitutive historical data for element type *EltType* at non-linear load step, *step*.
EltType.PHIST.*step*	Pointwise constitutive historical data for element type *EltType* at non-linear load step, *step*.

The GCP always creates three datasets on the database: MATL.PNTR, MATL.DATA, and FABRICATIONS, which contain pointers and data corresponding to the material and fabrication definitions. In addition, if there are no thermally dependent material properties or non-linear materials in the analysis the dataset, CONSTITUTIVE.STIFF is created storing the integrated constitutive stiffness for each individual fabrication definition.

The processor also creates external files containing historical data for those constitutive models that use such data. Two files are created: a converged data file containing historical data from the previous converged load step in a nonlinear analysis, and an iterative data file with results from the current iteration during the nonlinear analysis. If the GCP is run in stand-alone mode, the iterative and converged data files are named HISTORY.CONV and HISTORY.ITER. When the GCP is invoked by the element processor, the two files are named *EltType*.HIST.CONV and *EltType*.HIST.ITER, where *EltType* is the name of the element type being used, e.g. EX97.HIST.ITER.

Use of the UPDATE/ARCHIVE command (see Section 8.6), results in the creation of two datasets on the computational database containing historical data. The datasets are named *EltType*.EHIST.*step* and *EltType*.PHIST.*step*, where *step* is the step number in the nonlinear analysis. The dataset *EltType*.EHIST.*step* contains the element-based resultant historical data at

the element integration points, and *EltType*.PHIST.*step* contains the pointwise constitutive historical data at the layer integration points.

8.2.4 Limitations

There are two hard-coded array size limits that will affect users interested in shell fabrications. The number of layers in a single shell fabrication is presently hard-coded to a maximum of 100. The number of through-the-thickness integration points for a single layer is hard-coded to a maximum of 9.

The capability to interpolate material properties based on the value of the state parameters is not yet implemented in the GCP. Data associated with values of *npar1*, *npar2* > 1 (as documented in Subsection 8.4.1), will not be accessed in the current version; users should specify *npar2*, *npar1*(1:1) = 1.

Moisture-dependent computations are not implemented within the GCP.

8.2.5 Error Messages

Error messages within the GCP can be classified into two categories: user input errors and internal database access errors.

The GCP verifies that the user's input commands or number of data items are consistent with the command being executed. Reasonably descriptive messages are provided if an error of this type is encountered, typically directing the user to modify the input commands or data.

Internal database errors can be recognized by the first word being "Error" followed by information about the type of action being attempted (i.e. opening, closing, getting, or putting a particular type of data object). Typically if the user encounters one of these errors the COMET-AR computational database has not been properly initialized (i.e. the runstream was attempted on an existing database). If clearing the computational database and deletion of historical data files from the directory does not solve the error condition then the problem should be reported to the person in charge of the COMET-AR software.

8.3 Fabrication Definition

The fabrication definition commands allow the user to specify the geometry of the material structure at an element integration point (herein referred to as a fabrication point), and also serves as a pointer to the material properties to be used in the analysis. A unique fabrication type has been defined for each of the following kinematic idealizations: continuum, laminated shell, and beam.

Continuum fabrications represent the degenerate case: they have no geometric structure, since they correspond to a material point, and they simply require a pointer to material properties. Shell fabrications may be either homogeneous or partitioned into layers. Beam fabrications represent the cross-sectional properties of the section.

Fabrication definition is performed by a command subprocessor within the GCP. The commands used to invoke and exit the fabrication subprocessor are given below.

Command	Description
FABRICATION	Invoke fabrication subprocessor to define fabrications
ENDFAB	Exit fabrication subprocessor

All fabrication data may be input in either single or multiple fabrication input sessions. The commands for fabrication input for each kinematic idealization are given in Subsections 8.3.1 through 8.3.3.

8.3.1 Continuum Fabrication Definition

For a continuum idealization, a fabrication point is equivalent to a material point, hence each fabrication is limited to a unique material definition. For this reason the continuum fabrication consists of a fabrication identifier and a material identifier used to reference a database-resident set of material properties. The keyword-driven commands required to define a continuum fabrication are as follows.

```
SOLID
      FABID = fabid
      MATID= matid
END
```

The SOLID command is used to start the continuum fabrication definition, which is terminated with the END command. The SOLID and END commands are required for the definition of each individual continuum fabrication.

Parameter	Description
fabid	Continuum fabrication ID number
matid	Material ID number associated with the fabrication

An example defining two solid fabrications is given below, where fabrication number 1 references material 4 and fabrication number 2 references material 1.

```
FABRICATION
  SOLID
    FABID = 1
    MATID = 4
  END
  SOLID
    FABID = 2
    MATID = 1
  END
ENDFAB
```

8.3.2 Laminated Shell Fabrication Definition

For a laminated shell idealization (which includes monocoque plates as a special case) each fabrication point can contain a unique layup of materials. The keyword-driven commands used to define a shell fabrication are as follows.

```
SHELL
  FABID = fabid
  NLAYERS = nlayers
  [SCF = k1^2, k2^2 ]
  MATID[/SYMMETRIC] = matid_1, matid_2, ..., matid_nlayers
  THICKNESS[/SYMMETRIC] = t_1, t_2, ..., t_nlayers
  ANGLE[/SYMMETRIC] = θ_1, θ_2, ..., θ_nlayers
  INTPTS[/SYMMETRIC] = n_1, n_2, ..., n_nlayers
END
```

Input for the commands MATID, THICKNESS, ANGLE, and INTPTS can be reduced for a symmetric layup by adding the qualifier /SYMMETRIC to the command. When this option is used the number of layers must be even and the parameter values are entered beginning at the laminate midplane.

The SHELL command is used to start the shell fabrication definition, which is terminated with the END command. The parameters are described below.

Parameter	Description
fabid	Fabrication ID number
nlayers	Total number of layers in laminate fabrication.
k_1^2, k_2^2	Shear correction factors, defined in the fabrication coordinate system. Optional input; default values are $k_1^2, k_2^2 = 1.0$.
$matid_1, matid_2, \ldots, matid_{nlayers}$	Material ID number for each layer, *nlayers* entries.
$t_1, t_2, \ldots, t_{nlayers}$	Layer thickness for each layer, *nlayers* entries.
$\theta_1, \theta_2, \ldots, \theta_{nlayers}$	Orientation of material reference frame with respect to fabrication frame (θ has units of degrees), *nlayers* entries.
$n_1, n_2, \ldots, n_{nlayers}$	Number of integration points for each layer, *nlayers* entries. Optional input as described in detail below.

The GCP is presently hard-coded for a maximum of NLAYERS = 100.

The GCP performs a through-the-thickness integration to obtain the integrated stiffness coefficients or stress resultants for the laminate. The command INTPTS specifies the number of integration used within each layer and effectively controls the type of integration that is performed. For INTPTS = 1, a single integration point is chosen at the center of the layer, and the integrated quantity (stiffness or stress resultants) is assumed to be constant within the layer; this choice neglects bending effects in the laminate. For INTPTS = 2, the integration points are chosen at the top and bottom surface of the layer, and an exact integration is performed assuming a linear distribution of stress through the layer thickness. The results with INTPTS = 2 are equivalent to classical lamination theory. This choice is available to reduce the computational time for linear elastic materials, and should not be used for nonlinear materials. Finally, for INTPTS > 2, a repeated Simpson's integration rule is employed and the number of integration points must be odd. Presently, the GCP is hard-coded for a maximum of INTPTS = 9.

An example of an 8-layer symmetric aluminum-clad, $(0, \pm 60)_s$, P75 Graphite Epoxy laminate shell fabrication is given below.

```
FABRICATION
  SHELL
    FABID = 4 . Al-clad (0/+60/-60)s P75 gr/epoxy laminate
    NLAYERS = 8
    SCF = 0.833 0.833
    ANGLE/SYM = -60 60 0 0
    THICKNESS/SYM = 3@.005 .001
    MATID/SYM = 3@2 1
    INTPTS/SYM = 4@3 . optional input (default = 2)
  END
ENDFAB
```

8.3.3 Beam Fabrication Definition

The beam fabrication idealization is based on generalized properties used to define the cross-section. This method allows for only linear material behavior in beams with arbitrary cross-sections. The keyword-driven commands to define a beam fabrication are as follows.

```
BEAM
      FABID = fabid
      MATID = matid
      AREA = A
      MOMENTS = Iy, Iz, Iyz
      TORSION = J
      SHRCTR = cy, cz
      ECCEN = ey, ez
END
```

The BEAM command is used to start an individual beam fabrication definition, which is terminated with the END command. The parameters are described below.

Parameter	Description
fabid	Beam fabrication ID number
matid	Material ID number
A	Cross-sectional area
I_y, I_z, I_{yz}	Moments of inertia: $(I_y, I_z, I_{yz}) = \int_A (y^2, z^2, yz)dA$
J	Torsional rigidity of the beam cross-section
c_y, c_z	Shear center of the beam cross-section, measured from reference axes
e_y, e_z	Center of beam cross-section, measured from reference axes

8.4 Material Property Definition

The material property definition commands of the GCP allow the user to input the properties that characterize the behavior of the material for specific computational models. The GCP allows for user-defined constitutive models to be implemented in COMET-AR. This procedure is detailed in the GCP User's Manual. The material models listed below are now implemented in COMET-AR:

1) isotropic linear elasticity;

2) orthotropic linear elasticity;

3) isothermal mechanical sublayer plasticity.

The data input for these models is described in Subsections 8.4.4, 8.4.5, and 8.4.6.

Material property definition is performed by a command subprocessor within the GCP. The commands used to invoke and exit the material definition subprocessor are given below.

Command	Description
MATERIAL[/COMPUTATIONAL | /ARCHIVAL]	Invoke material definition subprocessor to define fabrications
ENDMAT	Exit material definition subprocessor

Input for the definition of these material properties is initiated by the command MATERIAL within the GCP processor environment. This is followed by the appropriate material property definition phrases, and terminated with the command ENDMAT. The command MATERIAL can have two optional qualifiers. If the qualifier /COMPUTATIONAL is specified, the GCP expects material property definitions to be input by the user for use in constructing the computational database. If the qualifier /ARCHIVAL is specified, the GCP will access an archival material database to retrieve the material property definition; alternatively, the MATERIAL/ARCHIVAL command can be used to archive a material property definition on the database. If the qualifier is omitted, the computational mode is assumed.

A general description of the material property data input is given in Subsection 8.4.1. Input of material properties from a pre-existing archival materials database to the GCP computational database is accomplished using the commands outlined in Subsection 8.4.2. Creation and maintenance of an archival material property database is detailed in Subsection 8.4.3.

8.4.1 Direct Material Property Definition

The general format for material property definition consists of a header line containing the material model name and general parameters, followed by one or more lines containing the actual material property data, as follows.

```
MATERIAL
    model_name matid, npar2, npar1(1:npar2)
    data
ENDMAT
```

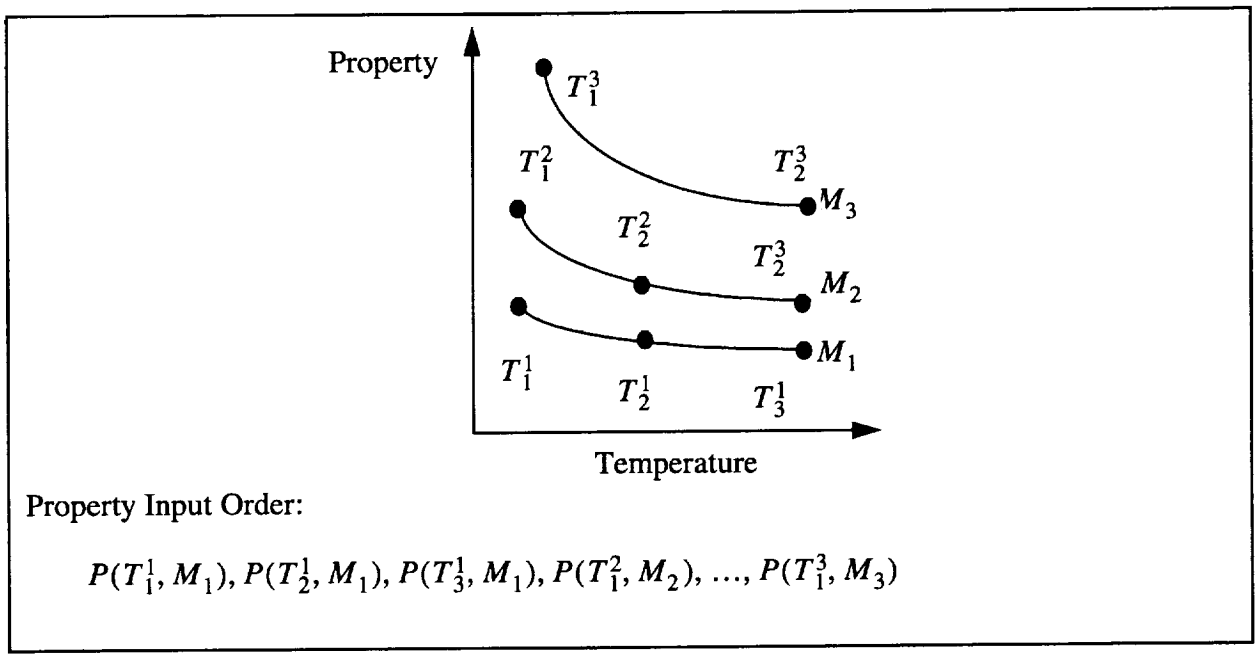
The input commands and parameters are described below.

Parameter	Description
model_name	Name of constitutive model (e.g., ISOEL, ORTEL, PLASTIC_WB).
matid	Material ID number to be use in analysis.
npar2	Number of specified values of state parameter 2; currently limited to *npar2*=1 in COMET-AR.
npar1(1:npar2)	Number of specified values of state parameter 1; currently limited to *npar1*(1:1) = 1 in COMET-AR.
data	Data for material model as outlined in Subsections 8.4.4–8.4.6.

The material property definition is designed to accommodate material properties that vary as a function of up to two state parameters, for example temperature and moisture. Input of material properties is repeated for each specified set of state parameters. The number of specified values of the second parameter is referred to as *npar2*; the number of specified values of the first parameter, at each specified value of the second parameter, is referred to as *npar1*. There may be a different number of specified values of the first parameter at each specified value of the second parameter.

For example, the user may specify 3 values of moisture (2nd parameter) and 3 values of temperature (1st parameter) at the first moisture value, 3 values of temperature at the second moisture value, and 2 values of temperature at the third moisture value (see Figure 8.4-1).The material property data is repeated such that the first state parameter varies first, the second state parameter is held constant (i.e., like a nested DO LOOP), the first parameter varies in the inner loop, and the second parameter varies in the outer loop. The actual values of these state parameters (e.g., temperature and moisture) are input on the same line with the associated material properties.

> The capability to interpolate material properties based on the value of the state parameters is not yet implemented in the GCP; data associated with values of *npar1*, *npar2*>1 will not be accessed in the current version.



Property Input Order:

$$P(T_1^1, M_1), P(T_2^1, M_1), P(T_3^1, M_1), P(T_1^2, M_2), \ldots, P(T_1^3, M_3)$$

Figure 8.4-1 Sample Variation of Property with Temperature and Moisture

8.4.2 Material Property Definition via Archival Database

Material property data may also be input from a predefined archival material database. To access archived material property definitions, first designate the archival database, then transfer the individual material constitutive model from the archival database to the current computational database. Commands to retrieve a material definition from an archival database are listed below.

```
MATERIAL/ARCHIVAL
   MATLIB ldi, libname
   MATL matid, dsname, model_name
ENDMAT
```

The MATLIB command is used to specify the material archival database, and the MATL command is used to enter a material definition into the current computational database. The parameters are described below.

Parameter	Description
ldi	Logical device index of archival material database.
libname	Name of archival material database.
matid	Material ID number to be used in analysis.
dsname	Dataset name in *libname* containing material definition.
model_name	Record name in *dsname* corresponding to constitutive model (e.g., ISOEL, ORTEL, PLASTIC_WB).

8.4.3 Creating and Maintaining Material Archival Libraries

There are three command options available for creating or maintaining a material archival library: CREATE, MODIFY, and DELETE. The commands act on datasets in the library; the particular action taken by each command is self-explanatory. The following command runstream illustrates their use for defining a new archival material library or operating on an existing library.

```
MATERIAL/ARCHIVAL
    {CREATE | MODIFY | DELETE}
        MATLIB ldi, libname
        MATNAM dsname
            model_name npar2, npar1(1:npar2)
            data
    {ENDCREATE | ENDMODIFY | ENDDELETE}
ENDMAT
```

The MATLIB command defines the archival database to use, and the MATNAM command identifies the dataset in *libname* which will be operated upon. The parameters are described below.

Parameter	Description
ldi	Logical device index of archival material database.
libname	Name of archival material database.
dsname	Dataset name in libname containing material definition.
model_name	Name of constitutive models described in Subsections 8.4.4–8.4.6.
npar2	Number of specified values of state parameter 2; currently limited to *npar*2=1 in COMET-AR.
npar1(1:npar2)	Number of specified values of state parameter 1; currently limited to *npar*1(1:1) =1 in COMET-AR.
data	Data for material model as outlined in Sections 8.4.4–8.4.6.

8.4.4 Linear Elastic Isotropic Material Property Definition

The material constitutive data expected for a temperature- and moisture-dependent linear elastic isotropic material are given below. The *model_name* and *data* items are entered under the MATERIAL command as described in Sections 8.4.1 through 8.4.3.

model_name	ISOEL
data	E, ν, ρ, α, β, T, M

The items in the data line are described below.

Parameter	Description
E	elastic modulus
v	Poisson's ratio
ρ	mass density
α	coefficient of thermal expansion
β	coefficient of hygroscopic expansion
T	reference temperature
M	reference moisture content

The definition of the material properties for an isotropic elastic material model of 6064-T4 Aluminum is shown below.

ISOEL 2 1 1	-- . .6061--T4 Aluminum
10.1E6	-- . Young's Modulus
0.29	-- . Poisson's Ratio
0.111	-- . Mass Density
12.0E-6	-- . CTE
0.0	-- . CHE (n/a)
75	-- . Temperature
0.0	-- . Moisture Content (n/a)

8.4.5 Linear Elastic Orthotropic Material Property Definition

The material constitutive data expected for a temperature- and moisture-dependent linear elastic orthotropic material are given below. The material property directions 1, 2, and 3 are referred to the material reference frame, which is oriented relative to the fabrication reference frame via the "angle" command for shell fabrications. The material and fabrication reference frames coincide for the beam and continuum fabrications. The fabrication reference frame is set according to the value of the FABRICATION DIRECTION command defined by the element processor (see Chapter 7).

The *model_name* and *data* items are entered under the material command as described in Sections 8.4.1 through 8.4.3.

model_name	ORTEL
data	$E_1, E_2, E_3, G_{12}, G_{13}, G_{23}, \nu_{12}, \nu_{13}, \nu_{23}, \rho, \alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3, T, M$

The items in the data line are described below.

Parameter	Description
E_1, E_2, E_3	elastic moduli
G_{12}, G_{13}, G_{23}	shear moduli
$\nu_{12}, \nu_{13}, \nu_{23}$	Poisson's ratios
ρ	mass density
$\alpha_1, \alpha_2, \alpha_3$	coefficients of thermal expansion
$\beta_1, \beta_2, \beta_3$	coefficients of hygroscopic expansion
T	reference temperature
M	reference moisture content

8.4.6 Mechanical Sublayer Plasticity Material Property Definition

The material constitutive data expected for the isothermal White-Besseling mechanical sublayer plastic material are given below. This model allows for the user to specify a pointwise effective stress-strain curve to simulate material behavior ranging from elastic/perfectly-plastic to strain-hardening plasticity (see Figure 8.4-2). The *model_name* and *data* items are entered under the MATERIAL command as described in Sections 8.4.1 through 8.4.3.

model_name	PLASTIC_WB
data	E, ν, ρ, α, n, $\sigma(i)$, $\varepsilon(i)$

The items in the data line are listed below.

Parameter	Description
E	elastic modulus
ν	Poisson's ratio
ρ	mass density
α	coefficient of thermal expansion
n	number of points in effective stress-strain curve
$\sigma, (i)$	effective stress for point i, $i = 1, n$
$\varepsilon, (i)$	effective strain for point i, $i = 1, n$

The mechanical sublayer model idealization of the uniaxial stress-strain curve of a hardening material is shown in Figure 8.4-2 for the case of a 3 sublayer model. The yield stress of each sublayer is found from:

$$\bar{\sigma}_k = E_1 \varepsilon_k \qquad k = 1, 2, 3$$

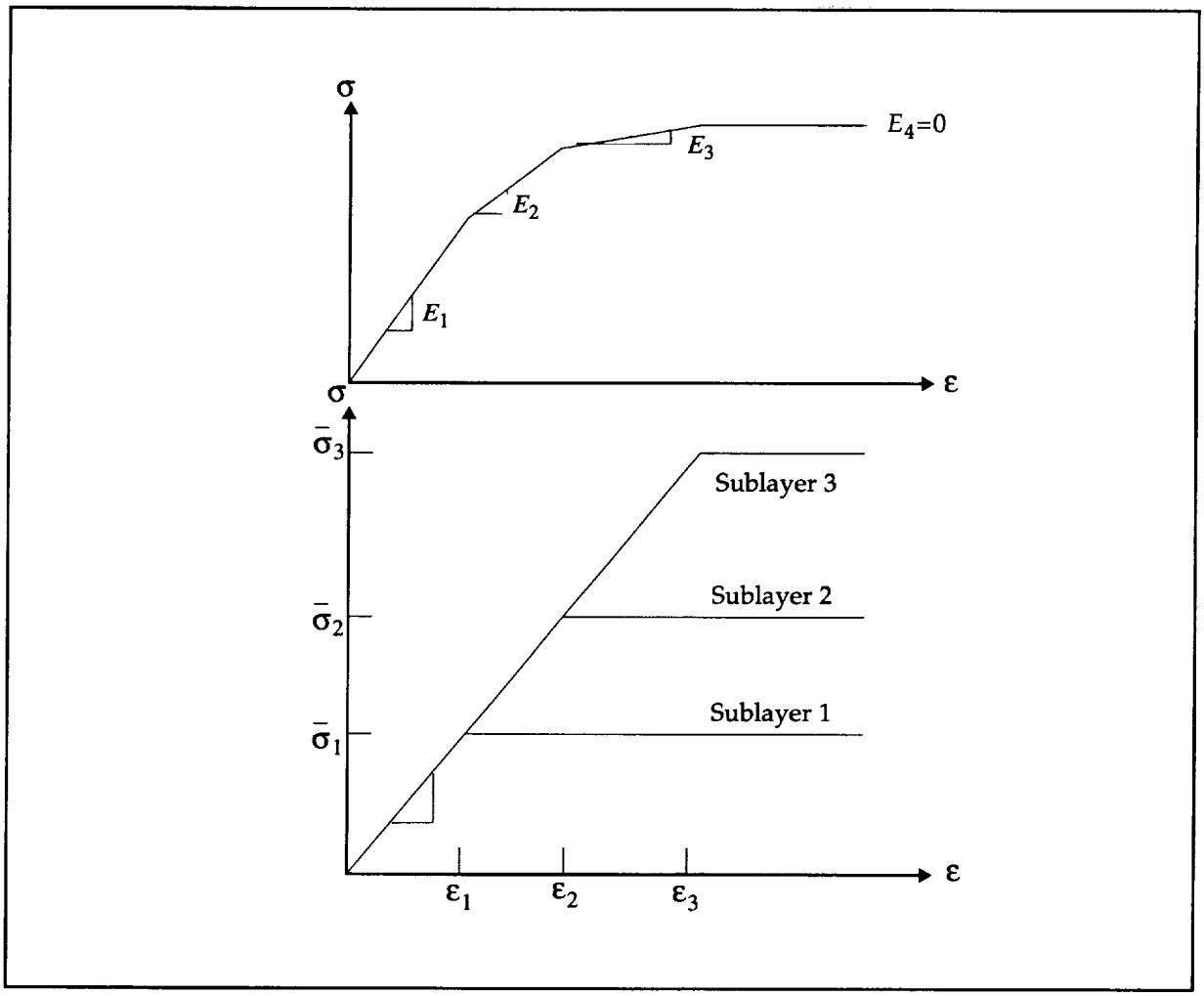


Figure 8.4-2 Mechanical Sublayer Model Idealization of the Uniaxial Stress-Strain Curve

An example of a command runstream for the definition of a material for the White-Besseling plasticity model is shown below.

```
MATERIAL
  PLASTIC_WB 1 1 1
    10000.              -- . Initial Elastic Modulus
    0.0                 -- . Poisson's Ratio
    0.0                 -- . Material Density
    0.0                 -- . Coefficient of Thermal Expansion
    3                   -- . Number of points on effective stress-strain curve
    10.0 15.0 17.0      -- . Effective Stress Values
    0.001 0.003 0.005   -- . Effective Strain Values
ENDMAT
```

8.5 Analysis Control

The analysis subprocessor is used to define the required information for controlling a pointwise constitutive analysis when using the GCP in stand-alone mode. This autonomous capability allows for pointwise constitutive analysis for continuum and beam and fabrications, as well as through-the-thickness integration to obtain stiffness and stress-resultant quantities for shell-fabrications. The GCP analysis capabilities are initiated by the following command phrase:

ANALYSIS = *analysis_type /kinematic_type*

where *analysis_type* identifies the type of analysis to be performed, and the */kinematic_type* qualifier indicates the kinematic assumptions to use in the analysis. The analysis command is followed by the user-selected analysis and load definition parameters, and the EXECUTE command is used to initiate the constitutive analysis.

Three stand-alone analysis types are available in the GCP: PROPERTIES, LINEAR, and NONLINEAR, summarized below and described in Sections 8.5.1, 8.5.2, and 8.5.3.

Analysis Type	Description
PROPERTIES	Computes the compliance properties for the particular *kinematic_type* chosen
LINEAR	Performs a linear stress analysis based on a user-described loading conditions
NONLINEAR	Performs a stress analysis at a series of user-prescribed load steps.

The analyses depend on the *kinematic_type* qualifier, the valid options for which are listed below.

kinematic_type	Description
1d	One dimensional continuum
2dstrs	Two dimensional plane-stress continuum
2dstrn	Two dimensional plane-strain continuum
2daxi	Two dimensional axisymmetric continuum
3d	Three dimensional continuum
c0shel	Shear-deformable shell
c1shel	Non-shear-deformable shell
c0beam	Shear-deformable beam
c1beam	Non-shear-deformable beam

The conjugate stress/strain quantities defined for the various kinematic models are listed in Table 8.5-1. For beam and shell fabrications, these quantities represent stress resultants and strain measures.

Table 8.5-1 Conjugate Stress-Strain Quantities for GCP Kinematic Types

kinematic_type	Stress (Resultant) Component	Strain (Measure) Component
1D	$\{\sigma_x\}$	$\{\varepsilon_x\}$
C0BEAM	$\{N_x, M_y, M_z, \tau, V_y, V_z\}$	$\{\varepsilon_x, \kappa_y, \kappa_z, \theta, \gamma_y, \gamma_z\}$
C1BEAM	$\{N_x, M_y, M_z, \tau\}$	$\{\varepsilon_x, \kappa_y, \kappa_z, \theta\}$
C0SHELL	$\{N_x, N_y, N_{xy}, M_x, M_y, M_{xy}, V_x, V_y\}$	$\{\varepsilon_x, \varepsilon_y, \varepsilon_{xy}, \kappa_x, \kappa_y, \kappa_{xy}, \gamma_x, \gamma_y\}$
C1SHELL	$\{N_x, N_y, N_{xy}, M_x, M_y, M_{xy}\}$	$\{\varepsilon_x, \varepsilon_y, \varepsilon_{xy}, \kappa_x, \kappa_y, \kappa_{xy}\}$
2DSTRESS	$\{\sigma_x, \sigma_y, \sigma_{xy}\}$	$\{\varepsilon_x, \varepsilon_y, \varepsilon_{xy}\}$
2DSTRAIN	$\{\sigma_x, \sigma_y, \sigma_{xy}\}$	$\{\varepsilon_x, \varepsilon_y, \varepsilon_{xy}\}$
2DAXISYM	$\{\sigma_r, \sigma_z, \sigma_{rz}, \sigma_\theta\}$	$\{\varepsilon_r, \varepsilon_z, \varepsilon_{rz}, \varepsilon_\theta\}$
3D	$\{\sigma_x, \sigma_y, \sigma_z, \sigma_{yz}, \sigma_{zx}, \sigma_{xy}\}$	$\{\varepsilon_x, \varepsilon_y, \varepsilon_z, \varepsilon_{yz}, \varepsilon_{zx}, \varepsilon_{xy}\}$

8.5.1 Property Analysis

The PROPERTIES command option evaluates the pointwise material properties for a constitutive model. The analysis returns the material compliance matrix for continuum and beam fabrications, and returns the integrated laminate compliance matrix for the shell fabrication. The following command is used to execute the properties analysis.

```
ANALYSIS = PROPERTIES /kinematic_type
EXECUTE
```

8.5.2 Linear Pointwise Stress Analysis

The LINEAR command option calculates the pointwise linear elastic stress state for a user-prescribed loading condition. Two types of loading can be defined for an analysis: mechanical and/or thermal loading. The following command runstream is used to execute the linear analysis.

```
ANALYSIS = LINEAR /kinematic_type
  TEMP input_cond, input_format, data, ref_temp
  LOAD
      LOAD_TYPE = STRAIN
      LOADS data
  EXECUTE
```

The TEMP command is used to define the temperature distribution for the fabrication. The *input_cond* code is used to specify whether the temperature data is an initial or current condition for the analysis, and can be set as follows.

input_cond	Description
1	Current condition for ANALYSIS = LINEAR, or initial condition for ANALYSIS = NONLINEAR.
2	Current condition for ANALYSIS = PROPERTIES, NONLINEAR.

Several assumed temperature distributions are available depending on the fabrication type. The continuum fabrication represents a material-point; hence, a constant temperature $\bar{T}$ is assumed for the point. The temperature distribution for the shell fabrication can be specified in the form of a constant temperature, a linear distribution, a quadratic distribution, a layer-wise linear distribution or a point-wise linear distribution. The through-the-thickness temperature distributions for the various shell formats are given by:

Format	Thermal Distribution	Valid Range
Constant	$T(z) = \bar{T}$	$-\frac{h}{2} < z < \frac{h}{2}$
Linear	$T(z) = \left(\frac{1}{2} - \frac{z}{h}\right)T_{bot} - \left(\frac{1}{2} + \frac{z}{h}\right)T_{top}$	$-\frac{h}{2} < z < \frac{h}{2}$
Quadratic	$T(z) = T_0 + T_1 z + T_2 z^2$	$-\frac{h}{2} < z < \frac{h}{2}$
Layer-wise linear	$T(z) = \left(\frac{z_{i+1} - z}{t_{i+1}}\right)T_i + \left(\frac{z - z_i}{t_{i+1}}\right)T_{i+1}$	$z_i < z < z_{i+1}$ $i = 0, 1, \ldots, nlayers$
Point-wise linear	$T(z) = \left(\frac{z_{i+1} - z}{z_{i+1} - z_i}\right)T_i + \left(\frac{z - z_i}{z_{i+1} - z_i}\right)T_{i+1}$	$z_i < z < z_{i+1}$ $i = 1, 2, \ldots, n$

where the thickness coordinate z is measured relative to the shell midsurface, t_i are the layer thicknesses, and h is the shell thickness. T_{top}, T_{bot} are the temperatures at $z = \pm h/2$, respectively; T_i are the layer interface temperatures for the layer-wise linear distribution, or can be specified at through-the-thickness coordinates z_i in the point-wise linear distribution.

Two temperature distributions are available for the beam fabrication: a constant temperature or a linear distribution, as shown below:

Format	Thermal Distribution
Constant	$T(y, z) = \bar{T}$
Linear	$T(y, z) = \bar{T} + y\frac{\partial T}{\partial y} + z\frac{\partial T}{\partial z}$

where y, z are measured from the reference axes.

The temperature distribution format is specified by an *input_format* code, which can take one of the following values.

input_format	Description
0	constant temperature
1	linear temperature distribution
2	quadratic temperature distribution
10	linear temperature distribution in each layer
20	linear temperature distribution between tabulated points

The required *data* for the TEMP command is listed below. The tabulated data for the point-wise linear format requires that n pairs of temperature and z-coordinate values be specified; the linear distribution for the beam fabrication requires that both temperatures and gradients be specified.

Format	Solid	Shell	Beam
Constant	$\bar{T}$	$\bar{T}$	$\bar{T}$
Linear	n/a	T_{top}, T_{bot}	$\bar{T}, \frac{\partial T}{\partial y}, \frac{\partial T}{\partial z}$
Quadratic	n/a	T_0, T_1, T_2	n/a
Layer-wise linear	n/a	$T_0, T_1, ..., T_{nlayers}$	n/a
Point-wise linear	n/a	$T_1, z_1, T_2, z_2, ..., T_n, T_z$	n/a

The last parameter in the TEMP command is *ref_temp*, which is the stress-free reference temperature.

The mechanical loading for the fabrication point is specified with the LOAD command. The GCP is currently limited to accepting mechanical loads in the form of strains for a continuum, or strain measures for beams and laminated shell fabrications. The LOADS command is used to enter the

prescribed (strain) data. The number and meaning of the data items should be consistent with the kinematic type as described in Table 8.5-1.

8.5.3 Nonlinear Pointwise Stress Analysis

The NONLINEAR command option calculates the pointwise stress state for a user-prescribed load history, which is specified at a number of steps. The following commands are used to execute the nonlinear analysis:

```
ANALYSIS = NONLINEAR /kinematic_type
    STEP n
        TEMP input_cond, input_format, data, ref_temp
        LOAD
            LOAD_TYPE = STRAIN
            LOADS data
EXECUTE
```

where the commands and parameters are identical to those described in Section 8.5.2, with the addition of STEP *n*, where *n* is the load step number. The commands STEP through LOADS are repeated as necessary to complete the load history definition.

8.5.4 Stand-Alone Analysis Examples

8.5.4.1 Material Property Analysis of an Elastic Orthotropic Laminated Shell

An example GCP runstream and the resulting output is presented for a pointwise material property (stiffness) analysis of a laminated shell made of both isotropic and orthotropic elastic material layers stacked in a symmetric layup.

The following runstream shows the GCP commands necessary to perform this analysis.

```
. ************************************************
. GCP Stand-alone Analysis Example #1
. Material property analysis of a laminated shell.
. ************************************************
.
*open 1 example1.dbc
.
. Input Laminated Shell Fabrication
.
fabrication
 shell
  fabid = 1
  nlayers = 8
  matid/sym =  3@1 2
  thick/sym = 0.005 3@0.0055
  angle/sym = 0. 45. -45. 90.
  intpts/sym = 4@3
 end
```

```
endfab
.
. Input Material Properties
.
material
 ortel 1 1 1
 42.0E6 1.2E6 1.2E6 .8E6 .8E6 .6E6 .3 .3 .3 0. 0. 0. 0. 0. 0. 0. 0. 0.
 isoel 2 1 1
 5.E5 0.45 0. 0. 0. 0. 0.
endmat
.
. Run Material Property Analysis
.
analysis properties /c1shel
execute
```

The output from the GCP for the above material property analysis is shown below. The result is the thickness-integrated composite laminate stiffness matrix, often known as the "ABD" matrix.

```
<CL> PUT_message,Commnt>
** BEGIN GCP  ** Using Dynamic Memory **
<CL> $root,L0001,C00001>*add example1.clp
<DM> OPEN,  Ldi:  1, File: example1.dbc , Attr: new, Block I/O

  SUMMARY OF SHELL FABRICATION NUMBER  1


  Composite Thickness =   4.3000E-02

  Layer no.      Layer          Layup angle    Thickness    Midplane distance
              Material no.    THETA (deg)         H                Z

      1             2             90.00       5.5000E-03      -1.8750E-02
      2             1            -45.00       5.5000E-03      -1.3250E-02
      3             1             45.00       5.5000E-03      -7.7500E-03
      4             1              0.00       5.0000E-03      -2.5000E-03
      5             1              0.00       5.0000E-03       2.5000E-03
      6             1             45.00       5.5000E-03       7.7500E-03
      7             1            -45.00       5.5000E-03       1.3250E-02
      8             2             90.00       5.5000E-03       1.8750E-02

<DM> OPEN,  Ldi:  2, File: HISTORY.CONV , Attr: new, Block I/O
<DM> CLOSE, Ldi:  2, File: HISTORY.CONV

 STIFFNESS
   0.6878E+06  0.2313E+06  0.1091E-10  0.6405E-03  0.2154E-03 -0.8527E-13
   0.2313E+06  0.2787E+06  0.1091E-10  0.2154E-03  0.2596E-03 -0.8527E-13
   0.1091E-10  0.1091E-10  0.2441E+06 -0.8527E-13 -0.8527E-13  0.2274E-03
   0.6405E-03  0.2154E-03 -0.8527E-13  0.3721E+02  0.2815E+02 -0.1299E+02
   0.2154E-03  0.2596E-03 -0.8527E-13  0.2815E+02  0.3380E+02 -0.1299E+02
  -0.8527E-13 -0.8527E-13  0.2274E-03 -0.1299E+02 -0.1299E+02  0.2893E+02

<CL> CSS exhausted
     ENDRUN called by CLIP
<DM> CLOSE, Ldi:  1, File: example1.dbc
```

8.5.4.2 Linear Stress Analysis of an Elastic Orthotropic Laminated Shell

An example GCP runstream and the resulting output is presented for a pointwise linear analysis of a laminated shell made of both isotropic and orthotropic elastic material layers stacked in a symmetric layup. The non-shear-deformable C^1 shell fabrication-point is subjected to a curvature-change κ_x of 0.001 in the "X-Z" plane, with all other strain components forced to remain zero. The following runstream shows the GCP commands necessary to perform this analysis.

```
. ************************************
. GCP Stand-alone Analysis Example #2
. Linear analysis of a laminated shell.
. ************************************
.
*open 1 example2.dbc
.
. Input Laminated Shell Fabrication
.
fabrication
 shell
  fabid = 1
  nlayers = 8
  matid/sym =  3@1 2
  thick/sym = 0.005 3@0.0055
  angle/sym = 0. 45. -45. 90.
  intpts/sym = 4@3
 end
endfab
.
. Input Material Properties
.
material
 ortel 1 1 1
 42.0E6 1.2E6 1.2E6 .8E6 .8E6 .6E6 .3 .3 .3 0. 0. 0. 0. 0. 0. 0. 0. 0.
 isoel 2 1 1
 5.E5 0.45 0. 0. 0. 0. 0.
endmat
.
. Run Linear Analysis
.
analysis linear /c1shel
  load
     load_type strain
     loads 0.0 0.0 0.0 0.001 0.0 0.0
execute
```

The output from the GCP stand-alone linear analysis consists of the strain and corresponding stress resultants at the shell fabrication point, as shown below.

```
 <CL> PUT_message,Commnt>
 ** BEGIN GCP  ** Using Dynamic Memory **
 <CL> $root,L0001,C00001>*add example2.clp
 <DM> OPEN,  Ldi:  1, File: example2.dbc , Attr: new, Block I/O

   SUMMARY OF SHELL FABRICATION NUMBER  1


   Composite Thickness =   4.3000E-02

   Layer no.      Layer        Layup angle   Thickness   Midplane distance
               Material no.    THETA (deg)       H              Z

        1            2            90.00      5.5000E-03     -1.8750E-02
        2            1           -45.00      5.5000E-03     -1.3250E-02
        3            1            45.00      5.5000E-03     -7.7500E-03
        4            1             0.00      5.0000E-03     -2.5000E-03
        5            1             0.00      5.0000E-03      2.5000E-03
        6            1            45.00      5.5000E-03      7.7500E-03
        7            1           -45.00      5.5000E-03      1.3250E-02
        8            2            90.00      5.5000E-03      1.8750E-02

 <DM> OPEN,  Ldi:  2, File: HISTORY.CONV , Attr: new, Block I/O
 <DM> CLOSE, Ldi:  2, File: HISTORY.CONV


  Linear Analysis
```

```
  STRAINS
    0.0000E+00  0.0000E+00  0.0000E+00  0.1000E-02  0.0000E+00  0.0000E+00

  STRESSES
    0.6405E-06  0.2154E-06  0.1665E-15  0.3721E-01  0.2815E-01 -0.1299E-01

 <CL> CSS exhausted
      ENDRUN called by CLIP
 <DM> CLOSE, Ldi:  1, File: example2.dbc
```

8.5.4.3 Linear Thermal Stress Analysis of an Orthotropic Laminated Shell

An example GCP runstream and the resulting output is presented for a pointwise linear analysis of a laminated shell made of both isotropic and orthotropic elastic material layers stacked in a symmetric layup. The non-shear-deformable C^1 shell fabrication-point is subjected to a linearly varying temperature through the thickness, with bottom and top surface temperatures of –1.0 and 1.0 degrees respectively. All mechanical strain components are constrained to remain zero. The following runstream shows the GCP commands necessary to perform this analysis.

```
. ***********************************************
. GCP Stand-alone Analysis Example #3
. Linear thermal analysis of a laminated shell.
. ***********************************************
.
*open 1 example3.dbc
.
. Input Laminated Shell Fabrication
.
fabrication
 shell
  fabid = 1
  nlayers = 8
  matid/sym =  3@1 2
  thick/sym = 0.005 3@0.0055
  angle/sym = 0. 45. -45. 90.
  intpts/sym = 4@3
 end
endfab
.
. Input Material Properties
.
material
   ortel 1 1 1
    42.0E6 1.2E6 1.2E6 .8E6 .8E6 .6E6 .3 .3 .3 0. .0001 .0001 0. 0. 0. 0. 0. 0.
   isoel 2 1 1
   5.E5 0.45 0.0001 0. 0. 0. 0.
endmat
.
. Run Linear Analysis
.
analysis linear /c1shel
  temp 1 1 1.0 -1.0 0.0
  load
     load_type strain
     loads 0.0 0.0 0.0 0.0 0.0 0.0
execute
.
```

The output from the GCP stand-alone linear analysis consists of the strain and corresponding stress resultants at the shell fabrication point, as shown below.

```
 <CL> PUT_message,Commnt>
 ** BEGIN GCP  ** Using Dynamic Memory **
```

```
 <CL> $root,L0001,C00001>*add example3.clp
 <DM> OPEN,  Ldi:  1, File: example3.dbc , Attr: new, Block I/O

   SUMMARY OF SHELL FABRICATION NUMBER  1


   Composite Thickness =   4.3000E-02

   Layer no.      Layer        Layup angle   Thickness   Midplane distance
              Material no.    THETA (deg)       H              Z

       1            2            90.00      5.5000E-03     -1.8750E-02
       2            1           -45.00      5.5000E-03     -1.3250E-02
       3            1            45.00      5.5000E-03     -7.7500E-03
       4            1             0.00      5.0000E-03     -2.5000E-03
       5            1             0.00      5.0000E-03      2.5000E-03
       6            1            45.00      5.5000E-03      7.7500E-03
       7            1           -45.00      5.5000E-03      1.3250E-02
       8            2            90.00      5.5000E-03      1.8750E-02

Echo of Temperature Command
  1  1    1.00000   -1.00000  0.
 <DM> OPEN,  Ldi:  2, File: HISTORY.CONV , Attr: new, Block I/O
 <DM> CLOSE, Ldi:  2, File: HISTORY.CONV


  Linear Analysis

   STRAINS
    0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00

   STRESSES
   -0.3809E-05 -0.1151E-05 -0.9745E-06 -0.2876E+00 -0.2717E+00  0.1209E+00

 <CL> CSS exhausted
      ENDRUN called by CLIP
 <DM> CLOSE, Ldi:  1, File: example3.dbc
```

8.5.4.4 Nonlinear Analysis of an Monocoque Shell

An example runstream and the resulting GCP output is presented for a pointwise nonlinear analysis of a monocoque shell. The shell fabrication-point is subjected to a load history of curvature-changes in the "X-Z" plane κ_x. The first load-step brings the point to the elastic limit of the shell. Subsequent load-steps continue the loading well into the plastic regime. The following runstream shows the GCP commands necessary to perform this analysis.

```
. ****************************************
. GCP Stand-alone Analysis Example #4
. Non-Linear analysis of a monocoque shell.
. ****************************************
.
*open 1 example4.dbc
.
. Input Monocoque Shell Fabrication
.
fabrication
   shell
     fabid = 1
     nlayers = 1
     matid = 1
     thick = 1.
     angle = 0.
     intpts = 5
   end
endfab
.
. Input Material Properties
.
material
```

```
 plastic_wb 1 1 1
 10000. 0.0 0.0 0.0 1 10.0 0.001
endmat
.
. Run Non-Linear Analysis
.
analysis nonlinear /c1shel
.
. Load Data For Step 1
.
  step 1

     load/incremental
       load_type  strain
       loads  0.0 0.0 0.0 0.002 0.0 0.0
.
. Load Data For Step 2
.
  step 2

     load/incremental
       load_type  strain
       loads  0.0 0.0 0.0 0.004 0.0 0.0
.
. Load Data For Step 3
.
  step 3

     load/incremental
       load_type  strain
       loads  0.0 0.0 0.0 0.006 0.0 0.0
.
. Load Data For Step 4
.
  step 4

     load/incremental
       load_type  strain
       loads  0.0 0.0 0.0 0.01 0.0 0.0
.
  execute
.
```

The output from the GCP stand-alone nonlinear analysis consists of the strain and corresponding stress resultants at the shell fabrication-point for each nonlinear load-step, as shown below.

```
 <CL> PUT_message,Commnt>
 ** BEGIN GCP  ** Using Dynamic Memory **
 <CL> $root,L0001,C00001>*add example4.clp
 <DM> OPEN,  Ldi:  1, File: example4.dbc , Attr: new, Block I/O

   SUMMARY OF SHELL FABRICATION NUMBER  1


   Composite Thickness =   1.0000E+00

   Layer no.      Layer        Layup angle   Thickness   Midplane distance
               Material no.    THETA (deg)       H              Z

       1            1             0.00      1.0000E+00      0.0000E+00

 <DM> OPEN,  Ldi:  2, File: HISTORY.CONV , Attr: new, Block I/O
 <DM> CLOSE, Ldi:  2, File: HISTORY.CONV


  STEP:    1

  STRAINS
    0.0000E+00  0.0000E+00  0.0000E+00  0.2000E-02  0.0000E+00  0.0000E+00
 <DM> OPEN,  Ldi:  2, File: HISTORY.CONV , Attr: old, Block I/O
 <DM> OPEN,  Ldi:  3, File: HISTORY.ITER , Attr: new, Block I/O
  STRESSES
```

```
  -0.1110E-15  0.0000E+00  0.0000E+00  0.1667E+01  0.1583E-07  0.0000E+00
<DM> CLOSE, Ldi:  2, File: HISTORY.CONV
<DM> CLOSE, Ldi:  3, File: HISTORY.ITER


 STEP:    2

 STRAINS
   0.0000E+00  0.0000E+00  0.0000E+00  0.4000E-02  0.0000E+00  0.0000E+00
<DM> OPEN,  Ldi:  2, File: HISTORY.CONV , Attr: old, Block I/O
<DM> OPEN,  Ldi:  3, File: HISTORY.ITER , Attr: new, Block I/O
 STRESSES
   0.1110E-15  0.0000E+00  0.0000E+00  0.2602E+01  0.2701E+00  0.0000E+00
<DM> CLOSE, Ldi:  2, File: HISTORY.CONV
<DM> CLOSE, Ldi:  3, File: HISTORY.ITER


 STEP:    3

 STRAINS
   0.0000E+00  0.0000E+00  0.0000E+00  0.6000E-02  0.0000E+00  0.0000E+00
<DM> OPEN,  Ldi:  2, File: HISTORY.CONV , Attr: old, Block I/O
<DM> OPEN,  Ldi:  3, File: HISTORY.ITER , Attr: new, Block I/O
 STRESSES
  -0.1110E-15 -0.5551E-16  0.0000E+00  0.2757E+01  0.7117E+00  0.0000E+00
<DM> CLOSE, Ldi:  2, File: HISTORY.CONV
<DM> CLOSE, Ldi:  3, File: HISTORY.ITER


 STEP:    4

 STRAINS
   0.0000E+00  0.0000E+00  0.0000E+00  0.1000E-01  0.0000E+00  0.0000E+00
<DM> OPEN,  Ldi:  2, File: HISTORY.CONV , Attr: old, Block I/O
<DM> OPEN,  Ldi:  3, File: HISTORY.ITER , Attr: new, Block I/O
 STRESSES
  -0.2220E-15  0.1110E-15  0.0000E+00  0.2866E+01  0.1180E+01  0.0000E+00
<DM> CLOSE, Ldi:  2, File: HISTORY.CONV
<DM> CLOSE, Ldi:  3, File: HISTORY.ITER
<CL> CSS exhausted
     ENDRUN called by CLIP
<DM> CLOSE, Ldi:  1, File: example4.dbc
```

8.6 Update Command

The GCP can store and access historical data if required by a constitutive material model, e.g., for plasticity analysis. Two external files are maintained: a converged data file containing historical data from the previous converged step in a nonlinear analysis; and an iterative data file with results from the previous iteration during the nonlinear analysis. The UPDATE command is used to copy the iterative data file to the converged file when a step converges in the nonlinear analysis.

UPDATE[/ARCHIVE LDI = *ldi* STEP = *step*]

When the optional qualifier ARCHIVE is specified, the historical data is also archived on the computational database identified by the parameter *ldi*; the dataset is named *EltType*.HIST.*step*, where *EltType* is the name of the element calling the GCP, and *step* is the load step number in the nonlinear analysis, e.g., EX97.HIST.*step*.

9 Smoothing Processors

9.1 Overview

Various smoothing processors implemented in COMET-AR are described in this chapter. These processors take basic finite element integration point data (e.g., stresses, strains, or strain energy densities) computed in the course of a standard finite element analysis, and compute globally smoothed (i.e., continuous) versions of this data at element integration points and/or nodes. In some cases, smoothed gradients of the basic quantities are also computed (e.g., processor SMT). The term "globally" is relative, as smoothing may be performed independently for different element groups. Such group partitioning is essential if physical discontinuities such as junctures, stiffness jumps, and concentrated loads, appear in the model, since smoothing should not be performed across such discontinuities.

The smoothing processors described in this chapter may be used in stand-alone mode for post-processing purposes, or as a basis for error estimation in adaptive mesh refinement. For this latter application, the user must select the smoothing processor and an appropriate error estimation processor (such as ERRSM) when invoking procedure AR_CONTROL (described in Chapter 4, *Adaptive Solution Procedures*).

A summary of currently available smoothing processors is given in Table 9.1-1.

Table 9.1-1 Outline of Chapter 9: Smoothing Processors

Section	Processor	Function
9.2	SMT	Smoothing based on Alex Tessler's algorithm.
9.3	SMZ	Smoothing based on the Zienkiewicz/Zhu smoothing algorithm.

The command language and database requirements for each of the above smoothing processors conform to common conventions. This is to facilitate their use by high-level procedures, such as AR_CONTROL, or in special-purpose user-written procedures.

9.2 Processor SMT (Tessler Smoothing)

9.2.1 General Description

Processor SMT computes element smoothed data (e.g., strains, stresses, and strain energies) employing the Tessler [1] global smoothing algorithm to obtain continuous fields over the problem domain, or specified subdomains (by element group).

The smoothing algorithm used by SMT is based on an approximate least-squares fit of a C^1 field to the finite element stress (or strain or strain-energy) solution which is typically piecewise continuous. The method is based on minimizing a functional involving discrete least-squares error plus a penalty constraint that ensures smoothness of the stress field. The result is a set of globally continuous stresses (or other quantity) plus continuous gradients in these quantities as well.

This smoothed data may be used either directly for post-processing, or by a generic smoothing-based error estimation processor (such as ERRSM) that may be used for adaptive mesh refinement (see Section 4.2, Procedure AR_CONTROL).

> Processor SMT is currently limited to planar (2D) structures.
>
> Smoothing should not be performed across physical discontinuities such as junctures, thickness jumps, material property jumps, or concentrated forces. Instead, the model should be partitioned into element groups that isolate the discontinuities, and the smoothing processor should be run independently for each element group.

9.2.2 Command Summary

Processor SMT follows standard COMET-AR command interface protocol. A summary of SMT commands is given in Table 9.2-1.

Table 9.2-1 Processor SMT Command Summary

Command Name	Function	Default Value
SET CONSTRAINT_SET	Specifies constraint-set number	1
SET ELEMENT_GROUP	Specifies subset of element groups to be smoothed as a single subdomain	0 (all)
SET ELEMENT_LIST	Specifies subset of element numbers to be smoothed within a single subdomain	0 (all)
SET ELEMENT_TYPE	Specifies subset of element types to be smoothed as a single subdomain	ALL

Table 9.2-1 Processor SMT Command Summary (Continued)

Command Name	Function	Default Value
SET GRADIENT_DATASET	Specifies the name for the smoothed gradients dataset (currently unused by SMT)	NONE
SET GRADIENT_FLAG	Sets the gradient processing flag	<false>
SET INPUT_DATASET	Sets the root name for the input dataset	STRESS
SET LDI	Sets the logical device index of the computational database library	1
SET LOAD_SET	Specifies the load-set number	1
SET MESH	Specifies the mesh number	0
SET OPTIONS	Sets selected smoothing options	BARLOW
SET OUTPUT_DATASET	Sets the root name for the output dataset	STRESS_SM
SET OUTPUT_LOCATIONS	Sets the location of the output data within each element (nodes, integration points, or both)	BOTH
SET SMOOTH_QUANTITY	Sets the quantity to be smoothed	STRESS
SET STEP	Sets the load step number	0
SMOOTH	Action command: compute smooth field	

9.2.3 Command Definitions

9.2.3.1 SMOOTH Command

This is the "go" command for processor SMT. It causes SMT to compute the smoothed nodal field values and output the element smoothed data as an EST dataset.

Command syntax:

SMOOTH

9.2.3.2 SET CONSTRAINT_SET Command

This command defines the constraint set number associated with the element and nodal data. This number should appear as the second cycle number in names of all datasets.

Command syntax:

SET CONSTRAINT_SET = *conset*

where

Parameter	Description
conset	Constraint set number (default value: 1)

9.2.3.3 SET ELEMENT_GROUP Command

This command defines the element group identity numbers for a group of elements that need to be processed by the ERRSM processor.

Command syntax:

SET ELEMENT_GROUP = *first:last:incr*
or
SET ELEMENT_GROUP = $g_1,g_2,\ldots,g_N$

where

Parameter	Description
first	First group ID to be processed
last	Last group ID
incr	Group ID increment
g_i	Group ID (default value: 0 — implies all groups)

9.2.3.4 SET ELEMENT_LIST Command

This command defines a subset of elements that need to be processed by the ERR*i* processors within the element group defined above.

Command syntax:

SET ELEMENT_LIST = *first:last:incr*
or
SET ELEMENT_LIST = $e_1,e_2,\ldots,e_n$

where

Parameter	Description
first	First element ID to be processed
last	Last element ID
incr	element ID increment
e_i	element ID (default value: 0 — implies all elements)

9.2.3.5 SET ELEMENT_TYPE Command

This command defines the subset of element types to be processed by the error estimation processor (e.g., ES1p, ES7p). This command is relevant only for linear static analysis.

Command syntax:

SET ELEMENT_TYPE = *element_type*

where

Parameter	Description
element_type	Element type name (default value: ALL)

9.2.3.6 SET GRADIENT_DATASET Command

This command defines the second component of the gradient dataset names. The full input dataset names are constructed as follows:

EltName.grad_name.step..mesh if *step* > 0

EltName.grad_name.ldset.conset.mesh if *step* = 0

Command syntax:

SET GRADIENT_DATASET = *grad_name*

where

Parameter	Description
grad_name	Second component of the gradient datasets name (default value: NONE)

9.2.3.7 SET GRADIENT_FLAG Command

This command sets the flag for the gradient processing option.

Command syntax:

SET GRADIENT_FLAG = *flag*

where

Parameter	Description
flag	Gradient processing flag (default value: 0 — implies false)

9.2.3.8 SET INPUT_DATASET Command

This command defines the second component of the input dataset names. The full input dataset names are constructed as follows:

EltName.in_name.step..mesh if *step* > 0

EltName.in_name.ldset.conset.mesh if *step* = 0

Command syntax:

SET INPUT_DATASET = *in_name*

where

Parameter	Description
in_name	Second component of the input datasets name (default value: STRESS)

9.2.3.9 SET LDI Command

This command defines the logical device index for the computational database.

Command syntax:

SET LDI = *ldi*

where

Parameter	Description
ldi	Logical device index. (default value: 1)

9.2.3.10 SET LOAD_SET Command

This command defines the load set number associated with the element and nodal data. This number should appear as the first cycle number in names of all datasets.

Command syntax:

SET LOAD_SET = *ldset*

where

Parameter	Description
ldset	Load set number (default value: 1)

9.2.3.11 SET MESH Command

This command defines the mesh number to be processed. This number should appear as the third cycle number in names of all datasets.

Command syntax:

SET MESH = *mesh*

where

Parameter	Description
mesh	Model mesh number (default value: 0)

9.2.3.12 SET OPTIONS Command

This command sets the Barlow stress point data rather then integration points data for computing the smoothed field.

Command syntax:

SET OPTIONS = *option*

where

Parameter	Description
option	Option value (default value: BARLOW)

9.2.3.13 SET OUTPUT_DATASET Command

This command defines the second component of the output dataset names. The full output dataset names are constructed as follows:

EltName.out_name.step..mesh if *step* > 0

EltName.out_name.ldset.conset.mesh if *step* = 0

Command syntax:

SET OUTPUT_DATASET = *out_name*

where

Parameter	Description
out_name	Second component of the output datasets name (default value: *in_name_SM*)

9.2.3.14 SET OUTPUT_LOCATIONS Command

This command defines the element location for the smoothed field output.

Command syntax:

SET OUTPUT_LOCATIONS = *location*

where

Parameter	Description
location	Element location of the smoothed field: NODES, INTEG_PTS, or BOTH (default value: BOTH)

9.2.3.15 SET SMOOTH_QUANTITY Command

This command defines the solution quantity to be smoothed.

Command syntax:

SET SMOOTH_QUANTITY = *quantity*

where

Parameter	Description
quantity	The solution quantity to be smoothed: STRESS, STRAIN, or STRAIN_ENERGY (default value: STRESS)

9.2.3.16 SET STEP Command

This command defines the solution step number associated with the element and nodal data. This number, if defined, should appear as the first cycle number in names of all datasets.

Command syntax:

SET STEP = *step*

where

Parameter	Description
step	Solution step number (default value: 0)

9.2.4 Database Input/Output

9.2.4.1 Input Datasets

A summary of input datasets required by Processor SMT is given below in Table 9.2-2.

Table 9.2-2 Processor SMT Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset
EltNam.INTERPOLATION...*mesh*	EIT	Element interpolation datasets
EltNam.DEFINITION...*mesh*	EDT	Element definition datasets
EltNam.in_name.ldset.conset.mesh —or— *EltNam.in_name.step..mesh*	EST	Element stress datasets. These must contain data evaluated at element integration points.
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate dataset

9.2.4.2 Output Datasets

A summary of output datasets created by Processor SMT is given in Table 9.2-3.

Table 9.2-3 Processor SMT Output Datasets

Dataset	Class	Contents
*EltNam.out_name.ldset.conset.mesh** —or— *EltNam.out_name.step..mesh* *	EST	Element smoothed stress datasets containing smoothed data at the required element locations.

*created dataset

9.2.5 Limitations

9.2.5.1 Partitioning Requirement

It may be necessary to partition the elements into element groups so that physical discontinuities (e.g., thickness jumps, point forces, or intersections in built-up structures) occur only on the boundary of such partitions. Otherwise, a meaningless smoothing of the physical discontinuity can result.

9.2.5.2 Element Type and Dimensionality

Currently, SMT is restricted to 2D (planar) elements and model geometries.

9.2.5.3 Common Solution Field Coordinate System

SMT interpolates, extrapolates, and accumulates element solution tensor contributions at each nodal point. These tensors are assumed to be defined in an appropriate reference coordinate system such that these type of operations are applicable. The user must verify that the element processors, ES*i*, are instructed to calculate strain and stress results in a common coordinate system for all elements that will be processed by SMT in a single SMOOTH command. This limitation is applicable only for tensor solution fields (i.e., STRESS and STRAIN). This limitation is not relevant to scalar solution quantities (i.e., STRAIN_ENERGY).

9.2.6 Error Messages

SMT contains extensive error checking. Most of the error messages printed by SMT are self-explanatory messages and aim to help the user correct mistakes. Some of the errors may occur at code levels below SMT (e.g., HDB, DB, GAL, etc.) and SMT describes those errors to the best of its ability.

The following is a summary of the error messages related to user interface problems as produced by SMT.

Index	Error Message	Cause	Recommended User Action
1	Invalid SMT command.	SMT encountered an unrecognized command.	Check the spelling of the command name, and refer to Command Descriptions in this section of the manual.
2	Invalid SET command.	SMT encountered an unrecognized SET command; i.e., the *object* in SET *object* is invalid.	Check the spelling of the SET option, and refer to Command Descriptions in this section of the manual.
3	Cannot open * dataset.	SMT could not open the named dataset.	1. Check the execution log file; look for error produced by processors prior to SMT execution. 2. Try to verify the dataset name using the HDBprt processor. 3. Make sure that all necessary input datasets are present in the database file.

9.2.7 Examples and Usage Guidelines

9.2.7.1 Example 1: Basic Operation

```
RUN SMT
                SET MESH    = 1
                SMOOTH
STOP
```

In this example, all default options are chosen except for the mesh number. This example will generate element smoothed stresses at both nodal and integration points locations using the Barlow point data for all active elements in the first mesh. The dataset name for the output dataset will be *EltNam*.STRESS_SM.1.1.1.

9.2.7.2 Example 2: Two Element Groups Partition

```
RUN SMT
                SET SMOOTH_QUANTITY        = STRAIN
                SET SMOOTH_LOCATIONS       = NODES
                SET INPUT_DATASET          = STRAIN_1
                SET MESH                   = 1
                SET ELEMENT_GROUP          = 1
                SMOOTH
                SET ELEMENT_GROUP          = 2
                SMOOTH
STOP
```

In this example, SMT is requested to smooth the elements strain fields and output the results at element nodal points only. The model is partitioned into two groups (due to physical discontinuity along the interface of these groups). For each group, a separate "go" command is issued to ensure that no smoothing will take place along the boundaries between the first and second groups of elements. The name for the output dataset will be *EltNam*.STRAIN_1_SM.1.1.

9.2.8 References

[1] Tessler, A., Riggs, H. R., and Macy, S. C., "A Variational Method for Finite Element Stress Recovery and Error Estimation," *AIAA Structures, Dynamics and Materials (SDM) Conference*, Paper AIAA-93-13844-CP, April, 1993.

9.3 Processor SMZ (Zienkiewicz/Zhu Smoothing)

9.3.1 General Description

Processor SMZ computes element smoothed data (e.g., strains, stresses, and strain energies) employing the Zienkiewicz-Zhu [1] global smoothing algorithm to obtain continuous fields over the problem domain or specified subdomains (by element group).

The smoothing algorithm used by SMZ is based on an approximate least-square fit of a C^o field to the finite element solution using the finite element displacement approximation space for the smoothed data field.

The smoothed field, f^{SM}, where f denotes stress, strain, or strain-energy density, is computed by SMZ is as follows:

$$f^{SM}(\mathbf{x}) = \sum_{a=1}^{Nen} N_a(\mathbf{x}) f_a^{SM} \tag{9.3-1}$$

where **x** denotes the local coordinates within an element, "a" is an element node number, Nen is the number of element nodes, N_a is the element shape function corresponding to element node "a," and f_a^{SM} is the globally smoothed (i.e., continuous) data at node "a," defined as follows:

$$f_a^{SM} = \frac{\int_\Omega N_a f^{FE} d\Omega}{\int_\Omega N_a^2 d\Omega} \tag{9.3-2}$$

In the above expression, the integrals are taken over all elements connected to node "a", which in effect is a weighted average of the basic finite element data, f^{FE}.

SMZ currently works in conjunction with any 2D (plate/shell) element implemented via the Generic Element Processor (i.e., ES*i* Processors). The output of SMZ is an element stress table (EST) data object, containing smoothed data at element integration points, via Equation 9.3-1, and/or at element nodes, via Equation 9.3-2. This smoothed data may be used either directly for post-processing, or by a generic smoothing-based error estimation processor (see ERRSM) for the assessment of element errors for a given mesh; the latter may be used for adaptive mesh refinement (see Section 4.2, Procedure AR_CONTROL).

9.3.2 Command Summary

Processor SMZ follows standard COMET-AR command interface protocol. A summary of SMZ commands is given in Table 9.3-1.

Table 9.3-1 Processor SMZ Command Summary

Command Name	Function	Default Value
SET CONSTRAINT_SET	Specifies constraint-set number	1
SET ELEMENT_GROUP	Specifies subset of element groups to be smoothed as a single subdomain	0 (all)
SET ELEMENT_LIST	Specifies subset of element numbers to be smoothed within a single subdomain	0 (all)
SET ELEMENT_TYPE	Specifies subset of element types to be smoothed as a single subdomain	ALL
SET GRADIENT_DATASET	Specifies the name for the smoothed gradients dataset (currently unused by SMZ)	NONE
SET GRADIENT_FLAG	Sets the gradient processing flag	<false>
SET INPUT_DATASET	Sets the root name for the input dataset	STRESS
SET LDI	Sets the logical device index of the computational database library	1
SET LOAD_SET	Specifies the load-set number	1
SET MESH	Specifies the mesh number	0
SET OPTIONS	Sets selected smoothing options	BARLOW
SET OUTPUT_DATASET	Sets the root name for the output dataset	STRESS_SM
SET OUTPUT_LOCATIONS	Sets the location of the output data within each element (nodes, integration points, or both)	BOTH
SET SMOOTH_QUANTITY	Sets the quantity to be smoothed	STRESS
SET STEP	Sets the load step number	0
SMOOTH	Action command: compute smooth field	

9.3.3 Command Definitions

9.3.3.1 SMOOTH Command

This is the "go" command for processor SMZ. It causes SMZ to compute the smoothed nodal field values using the Zienkiewicz-Zhu method and output the element smoothed data as an EST dataset.

Command syntax:

SMOOTH

9.3.3.2 SET CONSTRAINT_SET Command

This command defines the constraint set number associated with the element and nodal data. This number should appear as the second cycle number in names of all datasets.

Command syntax:

SET CONSTRAINT_SET = *conset*

where

Parameter	Description
conset	Constraint set number (default value: 1)

9.3.3.3 SET ELEMENT_GROUP Command

This command defines the element group identity numbers for a group of elements that need to be processed by the ERRSM processor.

Command syntax:

SET ELEMENT_GROUP = *first:last:incr*
or
SET ELEMENT_GROUP = $g_1, g_2, \ldots, g_N$

where

Parameter	Description
first	First group ID to be processed
last	Last group ID
incr	Group ID increment
g_i	Group ID (default value: 0 — implies all groups)

9.3.3.4 SET ELEMENT_LIST Command

This command defines a subset of elements that need to be processed by the ERRi processors within the element group defined above.

Command syntax:

SET ELEMENT_LIST = *first:last:incr*
or
SET ELEMENT_LIST = $e_1,e_2,...,e_n$

where

Parameter	Description
first	First element ID to be processed
last	Last element ID
incr	Element ID increment
e_i	Element ID (default value: 0 — implies all elements)

9.3.3.5 SET ELEMENT_TYPE Command

This command defines the subset of element types to be processed by the error estimation processor (e.g., ES1p, ES7p). This command is relevant only for linear static analysis.

Command syntax:

SET ELEMENT_TYPE = *element_type*

where

Parameter	Description
element_type	Element type name (default value: ALL)

9.3.3.6 SET GRADIENT_DATASET Command

This command defines the second component of the gradient dataset names. This command is not yet implemented. The full input dataset names are constructed as follows:

EltName.grad_name.step..mesh if *step* > 0

EltName.grad_name.ldset.conset.mesh if *step* = 0

Command syntax:

SET GRADIENT_DATASET = *grad_name*

where

Parameter	Description
grad_name	Second component of the gradient datasets name (default value: NONE)

9.3.3.7 SET GRADIENT_FLAG Command

This command sets the flag for the gradient processing option.

Command syntax:

SET GRADIENT_FLAG = *flag*

where

Parameter	Description
flag	Gradient processing flag (default value: 0 — implies false)

9.3.3.8 SET INPUT_DATASET Command

This command defines the second component of the input dataset names. The full input dataset names are constructed as follows:

EltName.in_name.step..mesh if *step* > 0

EltName.in_name.ldset.conset.mesh if *step* = 0

Command syntax:

SET INPUT_DATASET = *in_name*

where

Parameter	Description
in_name	Second component of the input datasets name (default value: STRESS)

9.3.3.9 SET LDI Command

This command defines the logical device index for the computational database.

Command syntax:

SET LDI = *ldi*

where

Parameter	Description
ldi	Logical device index. (default value: 1)

9.3.3.10 SET LOAD_SET Command

This command defines the load set number associated with the element and nodal data. This number should appear as the first cycle number in names of all datasets.

Command syntax:

SET LOAD_SET = *ldset*

where

Parameter	Description
ldset	Load set number (default value: 1)

9.3.3.11 SET MESH Command

This command defines the mesh number to be processed. This number should appear as the third cycle number in names of all datasets.

Command syntax:

SET MESH = *mesh*

where

Parameter	Description
mesh	Model mesh number (default value: 0)

9.3.3.12 SET OPTIONS Command

This command sets the Barlow stress point data rather then integration points data for computing the smoothed field.

Command syntax:

SET OPTIONS = *option*

where

Parameter	Description
option	Option value (default value: BARLOW)

9.3.3.13 SET OUTPUT_DATASET Command

This command defines the second component of the output dataset names. The full output dataset names are constructed as follows:

EltName.out_name.step..mesh if *step* > 0

EltName.out_name.ldset.conset.mesh if *step* = 0

Command syntax:

SET OUTPUT_DATASET = *out_name*

where

Parameter	Description
out_name	Second component of the output datasets name (default value: *in_name_SM*)

9.3.3.14 SET OUTPUT_LOCATIONS Command

This command defines the element location for the smoothed field output.

Command syntax:

SET OUTPUT_LOCATIONS = *location*

where

Parameter	Description
location	Element location of the smoothed field: NODES, INTEG_PTS, or BOTH (default value: BOTH)

9.3.3.15 SET SMOOTH_QUANTITY Command

This command defines the solution quantity to be smoothed.

Command syntax:

SET SMOOTH_QUANTITY = *quantity*

where

Parameter	Description
quantity	The solution quantity to be smoothed: STRESS, STRAIN, or STRAIN_ENERGY (default value: STRESS)

9.3.3.16 SET STEP Command

This command defines the solution step number associated with the element and nodal data. This number, if defined, should appear as the first cycle number in names of all datasets.

Command syntax:

SET STEP = *step*

where

Parameter	Description
step	Solution step number (default value: 0)

9.3.4 Database Input/Output

9.3.4.1 Input Datasets

A summary of input datasets required by Processor SMZ is given below in Table 9.3-2.

Table 9.2-2 Processor SMZ Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset
EltNam.INTERPOLATION...*mesh*	EIT	Element interpolation datasets
EltNam.DEFINITION...*mesh*	EDT	Element definition datasets

Table 9.2-2 Processor SMZ Input Datasets (Continued)

Dataset	Class	Contents
EltNam.in_name.ldset.conset.mesh —or— *EltNam.in_name.step..mesh*	EST	Element stress datasets. These must contain data evaluated at element integration points.
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate dataset

9.3.4.2 Output Datasets

A summary of output datasets created by Processor SMZ is given below in Table 9.3-3.

Table 9.2-3 Processor SMZ Output Datasets

Dataset	Class	Contents
*EltNam.out_name.ldset.conset.mesh** —or— *EltNam.out_name.step..mesh* *	EST	Element smoothed stress datasets containing smoothed data at the required element locations.

*created dataset

9.3.5 Limitations

9.3.5.1 Partitioning Requirement

It is necessary to partition the elements so that physical discontinuities (e.g., thickness jumps, point forces, or intersections in built-up structures) occur only on the boundary of element partitions. Otherwise, smoothing of the physical discontinuity may result.

9.3.5.2 Common Solution Field Coordinate System

SMZ interpolates, extrapolates, and accumulates element solution tensor contributions at each nodal point. These tensors are assumed to be defined in an appropriate reference coordinate system such that these type of operations are applicable. The user must verify that the element processors, ES*i*, are instructed to calculate strain and stress results in a common coordinate system for all elements that will be processed by SMZ in a single SMOOTH command. This limitation is applicable only for tensor solution fields (i.e., STRESS and STRAIN). This limitation is not relevant to scalar solution quantities (i.e., STRAIN_ENERGY).

9.3.6 Error Messages

SMZ contains extensive error checking. Most of the error messages printed by SMZ are self-explanatory messages and aim to help the user correct mistakes. Some of the errors may occur at code levels below SMZ (e.g., HDB, DB, GAL, etc.) and SMZ describes those errors to the best of its ability.

The following summarizes the error messages related to user interface problems as produced by SMZ:

Index	Error Message	Cause	Recommended User Action
1	Unknown *set variable name* encountered	SMZ encountered an unrecognized SET variable name.	Check the spelling of *variable name* in the CLIP procedure.
2	Unknown *command* encountered.	SMZ encountered an unrecognized COMMAND.	Check the spelling of the *command* in the CLIP procedure.
3	*Old/new dataset name* could not be opened.	SMZ could not open the named dataset.	1. Check the execution log file; look for error produced by processors prior to SMZ execution. 2. Try to verify the *dataset name* using the HDBprt processor. 3. Make sure that all input datasets are present in the database file.
4	*Dataset name* could not be closed.	SMZ could not close the named dataset.	1. Check the execution log file to look for errors previously produced by processor SMZ. 2. Verify that SMZ is the only processor accessing the database file (is ARGx being used in the same directory?).
5	*Dataset name* access problem encountered.	SMZ could not get/put an attribute from the dataset name table.	Verify that the named dataset contains attributes required by SMZ (e.g., EST does not contain data at integration points).

9.3.7 Examples and Usage Guidelines

9.3.7.1 Example 1: Basic Operation

```
RUN SMZ
                SET MESH    = 1
                SMOOTH
STOP
```

In the above example, all default options are chosen except for the mesh number. This example will generate element smoothed stresses at both nodal and integration points locations using the Barlow point data for all active elements in the first mesh.The dataset name for the output dataset will be *EltNam*.STRESS_SM.1.1.1.

9.3.7.2 Example 2: Two Element Groups Partition

```
RUN SMZ
                SET SMOOTH_QUANTITY          = STRAIN
                SET SMOOTH_LOCATIONS         = NODES
                SET INPUT_DATASET            = STRAIN_1
                SET MESH                     = 1
                SET ELEMENT_GROUP            = 1
                SMOOTH
                SET ELEMENT_GROUP            = 2
                SMOOTH
STOP
```

In the above example, SMZ is requested to smooth the elements strain fields and output the results at element nodal points only. The model is partitioned into two groups (due to physical discontinuity along the interface of these groups). For each group, a separate "go" command is issued to ensure that no smoothing will take place along the boundaries between the first and second groups of elements. The name for the output dataset will be *EltNam*.STRAIN_1_SM.1.1.

9.3.8 References

[1] Zienkiewicz, O. C. and Zhu, J. Z. "A Simple Error Estimator For Adaptive Procedure for Practical Engineering Analysis," *International Journal of Numerical Engineering*, Vol. 24, pp. 337-357, 1987.

10 Error Estimation Processors

10.1 Overview

In this chapter, various error estimation processors implemented in COMET-AR are described. These processors may be used either in stand-alone mode, or in conjunction with automated adaptive refinement procedures (see Chapter 4). By convention, the name of all error estimation processors begins with ERR; the rest of the name may be any unique alphanumeric string. A summary of currently available error estimation processors is given in Table 10.1-1.

Table 10.1-1 Outline of Chapter 10: Error Estimation Processors

Section	Processor	Function
10.2	ERR	Generic Error Estimator
10.3	ERR2	Zienkiewicz's strain-smoothing-based error estimates
10.4	ERR4	Levit's energy-smoothing-based error estimates
10.5	ERR6	Levit-modified version of ERR2
10.6	ERRa	Error accumulation processor
10.7	ERRSM	Error estimation post-processor for smoothing processors

The command language and database requirements for each of the above error estimation processors conform to common conventions. This is to facilitate their use by high-level procedures such as EST_ERR_1 and AR_CONTROL in the context of adaptive refinement.

10.2 Processor ERR (Generic Error Estimator)

10.2.1 General Description

Processors ERR*i* compute element error estimates using a variety of techniques which shall be individually discussed in the following sections of this chapter (ERR*i* is the generic name used to represent the name of any actually implemented error processor). To simplify the user interaction with the ERR*i* processors, a single generic user interface (which we refer to as ERR) is used as a cover for all error estimation processors.

This section will describe the common commands used in the generic error estimator user interface. Other information, such as database requirements, examples, and theoretical considerations are addressed under the individual ERR*i* processor sections.

10.2.2 Command Summary

Processors ERR*i* follow standard COMET-AR command interface protocol. A summary of valid commands is given in Table 10.2-1.

Table 10.2-1 Processor ERR*i* Command Summary

Command Name	Function	Default
SET CONSTRAINT_SET	Specifies constraint-set number for error estimation	1
SET ELEMENT_GROUP	Specifies subset of element groups for error estimates	0
SET ELEMENT_LIST	Specifies subset of element numbers for error estimates	0
SET ELEMENT_TYPE	Specifies subset of element types for error estimates	ALL
SET ERROR_TECHNIQUE	Specifies error estimation option	S
SET LDI	Specifies logical device index of computational database	1
SET LOAD_SET	Specifies load-set number for error estimation	1
SET MESH	Specifies mesh number for error estimation	0
SET STEP	Specifies load/time-step number for error estimation	0
ESTIMATE_ERRORS	Compute element error estimates; store in database	

10.2.3 Command Definitions

10.2.3.1 ESTIMATE ERRORS Command

This is the "go" command for processors ERR*i*. It causes ERR*i* to compute element errors for all or some of the elements in a specified mesh, and output them to an element error table (EET)

dataset *EltNam*.ERROR.*ldset.conset.mesh* (*EltNam* is the element name, *ldset* is the load set number, *conset* is the constraint set number, and *mesh* is the mesh number).

Command syntax:

ESTIMATE_ERRORS

10.2.3.2 SET CONSTRAINT_SET Command

This command defines the constraint set number associated with the element solution data for which error estimates are to be computed. This number should appear as the second cycle number in names of all element solution datasets, e.g., STRESS, STRAIN, and STRAIN_ENERGY. Relevant only for linear static analysis.

Command syntax:

SET CONSTRAINT_SET = *constraint_set*

where

Parameter	Description
constraint_set	Constraint set number (default value: 1)

10.2.3.3 SET ELEMENT_GROUP Command

This command defines the element group identity numbers for a group of elements that need to be processed by the ERR*i* processors for each of the element types specified.

Command syntax:

SET ELEMENT_GROUP = *first:last:incr*
or
SET ELEMENT_GROUP = $g_1,g_2,\ldots,g_N$

where

Parameter	Description
first	First group ID to be processed (default value: 0 — all groups)
last	Last group ID
incr	Group ID increment
g_i	Group ID

10.2.3.4 SET ELEMENT_LIST Command

This command defines a subset of elements that need to be processed by the ERR*i* processors within the element group defined above.

Command syntax:

SET ELEMENT_LIST = *first:last:incr*
or
SET ELEMENT_LIST = $e_1,e_2,...,e_n$

where

Parameter	Description
first	First element ID to be processed (default value: 0 — all elements)
last	Last element ID
incr	Element ID increment
e_i	Element ID

10.2.3.5 SET ELEMENT_TYPE Command

This command defines the subset of element types to be processed by the error estimation processor (e.g., ES1p, ES7p). Relevant only for linear static analysis.

Command syntax:

SET ELEMENT_TYPE = *element_type*

where

Parameter	Description
element_type	Element type name (default value: ALL)

10.2.3.6 SET ERROR_TECHNIQUE Command

This command defines the error technique to be used for estimating the solution errors (e.g., SMOOTHING, LOOK_AHEAD). Relevant only for linear static analysis.

Command syntax:

SET ERROR_TECHNIQUE = *error_technique/qualifier*

where

Parameter	Description
error_technique	Error technique (default value: SMOOTHING/BARLOW)

10.2.3.7 SET MESH Command

This command defines the mesh number associated with the model and solution data for which error estimates are to be computed. This number should appear as the third cycle number in names of all datasets (e.g., *EltNam*.ERROR.*ldset.conset.mesh*).

Command syntax:

SET MESH = *mesh*

where

Parameter	Description
mesh	Mesh number to be processed (default value: 0)

10.2.3.8 SET LDI Command

This command defines the logical device index for the central database.

Command syntax:

SET LDI = *ldi*

where

Parameter	Description
ldi	Logical device index (default value: 1)

10.2.3.9 SET LOAD_SET Command

This command defines the load set number associated with the element solution data for which error estimates are to be computed. This number should appear as the first cycle number in names of all element solution datasets (e.g., STRESS, STRAIN, and STRAIN_ENERGY). Relevant only for linear static analysis.

Command syntax:

SET LOAD_SET = *load_set*

where

Parameter	Description
load_set	Load set number. (default value: 1)

10.2.3.10 SET STEP Command

This command defines the solution step number associated with the element solution data for which error estimates are to be computed. This number should appear as the first cycle number in names of all element solution datasets (e.g., STRESS, STRAIN, and STRAIN_ENERGY). Relevant only for nonlinear static analysis.

Command syntax:

SET STEP = *step*

where

Parameter	Description
step	Solution step number. (default value: 0—implies linear analysis)

10.3 Processor ERR2 (Error Estimates: Stress Smoothing)

10.3.1 General Description

Processor ERR2 computes element error estimates employing the Zienkiewicz-Zhu [1] global smoothing algorithm to obtain a continuous strain field over the problem domain. The smoothed strain field is then compared to the original finite element approximation to yield an estimate of the element displacement error in terms of the strain energy norm.

The smoothing algorithm used by ERR2 is based on a least-square fit of a C^0 strain field to the finite element solution using the finite element displacement solution space for the smoothed strain field.

The definition of the element error estimate E_e computed by ERR2 is as follows:

$$E_e = \sqrt{\int_{\Omega_e} (\varepsilon^{SM} - \varepsilon^{FE})^T C (\varepsilon^{SM} - \varepsilon^{FE}) d\Omega}$$

$$\varepsilon^{SM} = \sum_{a=1}^{Nen} N_a \varepsilon_a^{SM}$$

$$\varepsilon_a^{SM} = \frac{\int_{\Omega} N_a \varepsilon^{FE} d\Omega}{\int_{\Omega} N_a^2 d\Omega}$$

ERR2 currently works in conjunction with any 2D (plate/shell) element implemented via the generic element processor (i.e., ES*i* processors), provided that the new AR-prototype version of the ES*i* shell is employed and that the element interpolation and extrapolation kernel routines (ES0IP and ES0XP) have been implemented for the particular element processor. The output of ERR2 is an element error table (EET) data object, which may be used for adaptive refinement (AR), or just as an assessment of element errors for a given mesh.

Processor ERR2 is normally invoked indirectly via procedure EST_ERR_1, which is called automatically by adaptive analysis procedures such as AR_CONTROL.

10.3.2 Command Summary

See Section 10.2 for the summary of the generic commands common to all error estimation processors.

10.3.3 Command Definitions

See Section 10.2 for the definition of the generic commands common to all error estimation processors.

10.3.4 Database Input/Output

10.3.4.1 Input Datasets

A summary of input datasets required by Processor ERR2 is given in Table 10.3-1.

Table 10.3-1 Processor ERR2 Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset.
EltNam.INTERPOLATION...*mesh*	EIT	Element interpolation datasets.
EltNam.DEFINITION...*mesh*	EDT	Element definition datasets.
EltNam.FABRICATION...*mesh*	EFT	Element fabrication datasets.
EltNam.STRAIN.*ldset.conset.mesh* *EltNam*.STRESS.*ldset.conset.mesh*	EST	Element strain and stress datasets. These must contain data evaluated at element integration points.
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate datasets.

10.3.4.2 Output Datasets

A summary of output datasets and attributes created by Processor ERR2 is given in Table 10.3-2.

Table 10.3-2 Processor ERR2 Output Datasets

Dataset	Class	Contents	
EltNam.ERROR.*ldset.conset.mesh**	EET	Element error datasets. The following element attributes are created.	
		Attribute	**Description**
		AbsErr	Absolute element error E_e
		Energy	Element strain-energy density: $U_e = \int_{\Omega_e} \hat{U}^{FE} d\Omega$ where $\hat{U}^{FE} = \frac{1}{2}\sigma \bullet \varepsilon$
		EngGrd	Element strain-energy density gradient: $\nabla\hat{U} = \frac{\hat{U}_{max} - \hat{U}_{min}}{\sqrt{A_e}}$ where A_e is the eleme8nt area.
		ErrRat	Element error ratio: $\tilde{E}_e = \frac{E_e/A_e}{\max_e (E_e/A_e)}$
		RelErr	Relative element error: $\hat{E}_e = \frac{E_e}{\sqrt{U_{tot}^{FE}/Nel}}$ where U_{tot}^{FE} is the total finite element strain energy integrated over all elements, and Nel is the total number of elements.

*—created dataset

10.3.5 Limitations

10.3.5.1 Partitioning Requirement

As with most smoothing-based error estimators, it is necessary to partition the elements so that physical discontinuities (e.g., thickness jumps, point forces, or intersections in built-up structures) occur only on the boundary of element partitions. Otherwise, excessive refinement in the vicinity of the physical discontinuity may result.

10.3.5.2 Common Strain Coordinate System

ERR2 interpolates, extrapolates, and adds element strain tensor contributions at each nodal point. These strain tensors are assumed to be defined in a consistent reference coordinate system such that these type of operations are applicable. The user must verify that the element processors, ES*i*, are instructed to calculate strain and stress results in a common coordinate system for all elements processed by ERR2 in a single ESTIMATE_ERRORS command.

10.3.6 Error Messages

ERR2 contains extensive checking. Most of the error messages printed by ERR2 are self-explanatory messages and aim to help the user correct mistakes. Some of the errors may occur at code levels below ERR2 (e.g., HDB, DB, GAL, etc.), and ERR2 describes those errors to the best of its ability. The following summarizes the error messages related to user interface problems as produced by ERR2.

Index	Error Message	Cause	Recommended User Action
1	Unknown *set variable name* encountered	ERR2 user interface cover encountered an unrecognized SET variable name.	Check the spelling of *variable name* in the CLIP procedure.
2	Unknown *command* encountered.	ERR2 user interface cover encountered an unrecognized COMMAND.	Check the spelling of *command* in the CLIP procedure.
3	*Old/new dataset name* could not be opened.	ERR2 could not open a certain dataset.	1. Check the execution log file; look for error produced by processors prior to ERR2 execution. 2. Try to verify the dataset name using the HDBprt processor. 3. Make sure that all input datasets are present in the database file.
4	*Dataset name* could not be closed.	ERR2 could not close a certain Dataset.	1. Check the execution log file, look for errors previously produced by processor ERR2. 2. Verify that ERR2 is the only processor accessing the database file (is ARGx being used in the same directory?).
5	*Dataset name* access problem encountered.	ERR2 could not get/put an attribute from the dataset name table.	Verify that the particular dataset contains attributes required by ERR2 (e.g., EST contain nontrivial data at integration point).

10.3.7 Examples and Usage Guidelines

10.3.7.1 Example 1: Basic Operation

```
RUN ERR2
                SET MESH            = 1
                ESTIMATE ERRORS
STOP
```

In this example, all default options are chosen except for the mesh number.

10.3.7.2 Example 2: Two Element Groups Partition

```
RUN ERR2
                SET MESH            = 1
                SET ELEMENT_GROUP   = 1
                ESTIMATE ERRORS
                SET ELEMENT_GROUP   = 2
                ESTIMATE ERRORS
STOP
```

In this example, all default options are chosen except for the mesh number and element group. For each group, a separate "go" command is issued to ensure that no smoothing will take place along the boundaries between the first and second groups of elements.

10.3.8 References

[1] Zienkiewicz, O. C. and Zhu, J. Z. "A Simple Error Estimator For Adaptive Procedure for Practical Engineering Analysis," *International Journal of Numerical Engineering*, Vol. 24, pp. 337-357, 1987.

10.4 Processor ERR4 (Error Estimates: Energy Smoothing)

10.4.1 General Description

Processor ERR4 computes element error estimates employing element-computed strain energy densities, and a smoothing-based projection technique. Like Processor ERR2, ERR4 uses a Zienkiewicz-type [1] global smoothing algorithm to obtain a continuous strain energy field over the problem domain. The smoothed strain energy field is then compared to the original finite-element approximation to yield an estimate of the element displacement error in terms of the strain energy norm. ERR4 is different from ERR2 in that the smoothing algorithm is applied to the square root of the strain energy density field instead of the strain field. The error expression itself is modified in ERR4 to involve these square roots directly.

The error estimation algorithm used by ERR4 leads to a significant increase in implementation simplicity and efficiency due to smoothing a scalar quantity (strain energy density) rather than a tensor quantity (strain). Unlike processor ERR2, ERR4 is applicable to arbitrary structural configurations including built-up shell structures (e.g., stiffened shells) and to models involving different type of elements (e.g., shells and beams).

The definition of the element error estimate E_e computed by ERR4 is as follows.

$$E_e = \sqrt{\int_{\Omega_e} \left(\sqrt{\hat{U}_e^{SM}} - \sqrt{\hat{U}_e^{FE}}\right)^2 d\Omega}$$

$$\hat{U}^{FE} = \sigma_{ij}\varepsilon_{ij}$$

$$\sqrt{\hat{U}_e^{SM}} = \sum_{a=1}^{Nen} N_a \sqrt{\hat{U}_a^{SM}}$$

$$\sqrt{\hat{U}_a^{SM}} = \frac{\int_{\Omega} N_a \sqrt{\hat{U}^{FE}} d\Omega}{\int_{\Omega} N_a^2 d\Omega}$$

ERR4 currently works in conjunction with any 2D (plate/shell) element implemented via the generic element processor (i.e., ES*i* processors), provided that the new AR-prototype version of the ES*i* shell is employed and that the element interpolation and extrapolation kernel routines (ES0IP and ES0XP) have been implemented for the particular element processor. The output of ERR4 is an element error table (EET) data object, which may be used for adaptive refinement (AR), or just as an assessment of element errors for a given mesh.

Processor ERR4 is normally invoked indirectly via procedure EST_ERR_1, which is called automatically by adaptive analysis procedures such as AR_CONTROL.

10.4.2 Command Summary

See Section 10.2 for the summary of the generic commands common to all error estimation processors.

10.4.3 Command Definitions

See Section 10.2 for the definition of the generic commands common to all error estimation processors.

10.4.4 Database Input/Output

10.4.4.1 Input Datasets

A summary of input datasets required by Processor ERR4 is given below in Table 10.4-1.

Table 10.4-1 Processor ERR4 Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset.
EltNam.INTERPOLATION...*mesh*	EIT	Element interpolation datasets.
EltNam.DEFINITION...*mesh*	EDT	Element definition datasets.
EltNam.STRAIN_ENERGY.*ldset.conset.mesh*	EST	Element strain energy dataset. These must contain strain energy densities at element integration points.
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate datasets.

10.4.4.2 Output Datasets

A summary of output datasets and attributes created by Processor ERR4 is given in Table 10.4-2.

Table 10.4-2 Processor ERR4 Output Datasets

Dataset	Class	Contents	
EltNam.ERROR.*ldset.conset.mesh**	EET	Element error datasets. The following element attributes are created.	
		Attribute	**Description**
		AbsErr	Absolute element error E_e
		Energy	Element strain-energy density: $U_e = \int_{\Omega_e} \hat{U}^{FE} d\Omega$ where $\hat{U}^{FE} = \frac{1}{2}\sigma \bullet \varepsilon$
		EngGrd	Element strain-energy density gradient: $\nabla\hat{U} = \frac{\hat{U}_{max} - \hat{U}_{min}}{\sqrt{A_e}}$ where A_e is the element area.
		ErrRat	Element error ratio: $\tilde{E}_e = \frac{E_e/A_e}{\max_e (E_e/A_e)}$
		RelErr	Relative element error: $\hat{E}_e = \frac{E_e}{\sqrt{U_{tot}^{FE}/Nel}}$ where U_{tot}^{FE} is the total finite element strain energy integrated over all elements, and Nel is the total number of elements.

*—created dataset

10.4.5 Limitations

10.4.5.1 Effectivity

The error estimates computed by processor ERR4 tend to underestimate rather than overestimate the actual error.

10.4.5.2 Change of Sign Errors

Processor ERR4 employs strain energy density for measuring the errors. The square root of the strain energy is equivalent to the weighted norm of the stress tensor and as such is insensitive to sign changes in any stress component. As a result, ERR4 will produce significant spurious errors in areas of the model in which a change in sign of a dominant stress component occurs.

10.4.5.3 Partitioning Due to Change in Shell Thicknesses

Processor ERR4 smooths the square root of the strain energy density (a scalar quantity) and thus does not require any special partitioning for structures in which the physical strain energy fields are continuous. For true shell elements, the ES*i* processors computes the strain energy densities based on resultant stresses and thus the strain energy densities are per unit element area (e.g., contain thickness information).

For structures containing thickness discontinuities, or any physical strain energy discontinuities, the model should be partitioned into groups of element such that ERR4 will preserve the discontinuity in the smoothed field solution. This will prevent generation of spurious errors along the physical discontinuities.

10.4.6 Error Messages

ERR4 contains extensive error checking. Most of the error messages printed by ERR4 are self-explanatory messages and aim to help the user correct his mistakes. Some of the errors may occur at code levels below ERR4 (e.g., HDB, DB, GAL, etc.), and ERR4 describes those errors to the best of its ability.

The following summarizes the error messages related to user interface as produced by ERR4.

Index	Error Message	Cause	Recommended User Action
1	Unknown *set variable name* encountered	ERR4 user interface cover encountered an unrecognized SET variable name.	Check spelling of *variable name* in CLIP procedure.
2	Unknown *command* encountered.	ERR4 user interface cover encountered an unrecognized command.	Check spelling of *command* in CLIP procedure.
3	*Old/new dataset name* could not be opened.	ERR4 could not open a certain dataset.	1. Check the execution log file; look for error produced by processors prior to ERR4 execution. 2. Verify the dataset name using HDBprt processor. 3. Make sure that all input datasets are present in the database file.
4	*Dataset name* could not be closed.	ERR4 could not close a certain dataset.	1. Check the execution log file; look for errors previously produced by processor ERR4. 2. Verify that ERR4 is the only processor accessing database file (is ARGx used in the same directory?).
5	*Dataset name* access problem encountered.	ERR4 could not get/put an attribute from the dataset name table.	Verify that the dataset contains attributes required by ERR4 (e.g., EST contains nontrivial data at integration point).

10.4.7 Examples and Usage Guidelines

10.4.7.1 Example 1: Basic Operation

```
RUN ERR4
              SET MESH              = 1
              ESTIMATE ERRORS
STOP
```

In this example, all default options are chosen except for the mesh number.

10.4.7.2 Example 2: Two Element Groups Partition

```
RUN ERR4
              SET MESH              = 1
              SET ELEMENT_GROUP     = 1
              ESTIMATE ERRORS
              SET ELEMENT_GROUP     = 2
              ESTIMATE ERRORS
STOP
```

In this example, all default options are chosen except for the mesh number and element group. For each group, a separate "go" command is issued to ensure that no smoothing will take place along the boundaries between the first and second groups of elements.

10.4.8 References

[1] Stanley, G., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR: Adaptive Refinement (AR) Manual*, 1991.

10.5 Processor ERR6 (Error Estimates: Stress Smoothing)

10.5.1 General Description

Processor ERR6 computes element error estimates employing the Zienkiewicz-Zhu [1] global smoothing algorithm to obtain a continuous strain field over the problem domain. The smoothed strain field is then compared to the original finite-element approximation to yield an estimate of the element displacement error in terms of the strain energy norm.

The smoothing algorithm used by ERR6 is based on a least-square fit of a C^o strain field to the finite element solution using the finite element displacement solution space for the smoothed strain field. The main difference between processors ERR2 and ERR6 is in the definition of the error norm. ERR6 formulation is based on the theorem "the energy error is equal to the error in energies," namely:

$$E^2 = \int_{\Omega} (\varepsilon^{Exact} - \varepsilon^{FE})^T C (\varepsilon^{Exact} - \varepsilon^{FE}) d\Omega = U^{Exact} - U^{FE}$$

where

$$U = \int_{\Omega} \varepsilon^T C \varepsilon d\Omega$$

The definition of the element error estimate E_e computed by ERR6 is as follows.

$$E_e = \sqrt{U_e^{SM} - U_e^{FE}}$$

$$U_e^{SM} = \int_{\Omega_e} (\varepsilon^{SM})^T C \varepsilon^{SM} d\Omega$$

$$\varepsilon^{SM} = \sum_{a=1}^{Nen} N_a \varepsilon_a^{SM}$$

$$\varepsilon_a^{SM} = \frac{\int_{\Omega} N_a \varepsilon^{FE} d\Omega}{\int_{\Omega} N_a^2 d\Omega}$$

ERR6 currently works in conjunction with any 2D (plate/shell) element implemented via the generic element processor (i.e., ES*i* processors), provided that the new AR-prototype version of the ES*i* shell is employed and that the element interpolation and extrapolation kernel routines (ES0IP and ES0XP) have been implemented for the particular element processor. The output of ERR6 is an element error table (EET) data object, which may be used for adaptive refinement (AR), or just as an assessment of element errors for a given mesh.

Processor ERR6 is normally invoked indirectly via procedure EST_ERR_1, which is called automatically by adaptive analysis procedures such as AR_CONTROL.

10.5.2 Command Summary

See Section 10.2 for the summary of the generic commands common to all error estimation processors.

10.5.3 Command Definitions

See Section 10.2 for the definition of the generic commands common to all error estimation processors.

10.5.4 Database Input/Output

10.5.4.1 Input Datasets

A summary of input datasets required by Processor ERR6 is given in Table 10.5-1.

Table 10.5-1 Processor ERR6 Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset.
EltNam.INTERPOLATION...*mesh*	EIT	Element interpolation datasets.
EltNam.DEFINITION...*mesh*	EDT	Element definition datasets.
EltNam.FABRICATION...*mesh*	EFT	Element fabrication datasets.
EltNam.STRAIN.*ldset.conset.mesh* *EltNam*.STRESS.*ldset.conset.mesh*	EST	Element strain and stress datasets. Must contain data evaluated at element integration points.
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate datasets.

10.5.4.2 Output Datasets

A summary of output datasets and attributes created by Processor ERR6 is given in Table 10.5-2.

Table 10.5-2 Processor ERR6 Output Datasets

Dataset/Attribute	Class	Contents	
EltNam.ERROR.*ldset.conset.mesh**	EET	Element error datasets. These element attributes are created.	
		Attribute	**Description**
		AbsErr	Absolute element error E_e
		Energy	Element strain-energy density: $U_e = \int_{\Omega_e} \hat{U}^{FE} d\Omega$ where $\rho^{FE} = \sigma:\varepsilon$
		EngGrd	Element strain-energy density gradient: $\nabla\hat{U} = \frac{\hat{U}_{max} - \hat{U}_{min}}{\sqrt{A_e}}$ where A_e is the element area.
		ErrRat	Element error ratio: $\tilde{E}_e = \frac{E_e/A_e}{\max_e (E_e/A_e)}$
		RelErr	Relative element error: $\hat{E}_e = \frac{E_e}{\sqrt{U_{tot}^{FE}/Nel}}$ where U_{tot}^{FE} is the total finite element strain energy integrated over all elements, and Nel is the total number of elements.

*—created dataset

10.5.5 Limitations

10.5.5.1 Partitioning Requirement

As with most smoothing-based error estimators, it is necessary to partition the elements so that physical discontinuities (e.g., thickness jumps, point forces, or intersections in built-up structures)

occur only on the boundary of element partitions. Otherwise excessive refinement in the vicinity of the physical discontinuity may result.

10.5.5.2 Common Strain Coordinate System

ERR2 interpolates, extrapolates, and adds element strain tensor contributions at each nodal point. These strain tensors are assumed to be defined in a consistent reference coordinate system such that these type of operations are applicable. The user must verify that the element processors, ES*i*, are instructed to calculate strain and stress results in a common coordinate system for all elements processed by ERR2 in a single ESTIMATE_ERRORS command.

10.5.6 Error Messages

ERR6 contains extensive checking. Most of the error messages printed by ERR6 are self-explanatory messages and aim to help the user correct mistakes. Some of the errors may occur at code levels below ERR6 (e.g., HDB, DB, GAL, etc.) and ERR2 describes those errors to the best of its ability. The following summarizes the error messages related to user interface problems as produced by ERR6:

Index	Error Message	Cause	Recommended User Action
1	Unknown *set variable name* encountered	ERR6 user interface cover encountered an unrecognized SET variable name.	Check spelling of *variable name* in CLIP procedure.
2	Unknown *command* encountered.	ERR6 user interface cover encountered an unrecognized command.	Check spelling of *command* in CLIP procedure.
3	*Old/new dataset* name could not be opened.	ERR6 could not open a certain dataset.	1. Check the execution log file; look for error produced by processors prior to ERR6 execution. 2. Try to verify the particular Dataset using the HDBprt processor. 3. Make sure that all input datasets are present in the database file.
4	*Dataset name* could not be closed.	ERR6 could not close a certain dataset.	1. Check the execution log file; look for errors previously produced by processor ERR6. 2. Verify that ERR6 is the only processor accessing the database file (is ARGx being used in the same directory?).
5	*Dataset name* access problem encountered.	ERR6 could not get/put an attribute from the dataset name table.	Verify that dataset contains attributes required by ERR6 (e.g., EST contains nontrivial data at integration point).

10.5.7 Examples and Usage Guidelines

10.5.7.1 Example 1: Basic Operation

```
RUN ERR6
              SET MESH              = 1
              ESTIMATE ERRORS
STOP
```

In this example, all default options are chosen except for the *mesh* number.

10.5.7.2 Example 2: Two Element Groups Partition

```
RUN ERR6
              SET MESH              = 1
              SET ELEMENT_GROUP     = 1
              ESTIMATE ERRORS
              SET ELEMENT_GROUP     = 2
              ESTIMATE ERRORS
STOP
```

In this example, all default options are chosen except for the mesh number and element group. For each group, a separate "go" command is issued to ensure that no smoothing will take place along the boundaries between the first and second groups of elements.

10.5.8 References

[1] Zienkiewicz, O. C., and Zhu, J. Z., "A Simple Error Estimator For Adaptive Procedure for Practical Engineering Analysis," *International Journal of Numerical Engineering*, Vol. 24, pp. 337-357, 1987.

10.6 Processor ERRa (Error Accumulator)

10.6.1 General Description

Processor ERRa is used to compute total errors in cases when smoothing-based error estimators (e.g., ERR2, ERR4, and ERR6) force the user to employ partitioning in estimating element errors.

When model partitioning is used, each invocation of an ERR*i* processor computes the errors (including global errors) only for user-specified elements (see Section 10.2 for user commands for partitioning the model). The main function of ERRa is to accumulate intermediate values and compute the total model errors, total strain energy, relative element error, and element errors ratios based on all elements data.

Processor ERRa is normally invoked indirectly via procedure EST_ERR_1, called automatically by adaptive analysis procedures such as AR_CONTROL.

10.6.2 Command Summary

Processors ERRa follow standard COMET-AR command interface protocol. A summary of ERRa commands is given in Table 10.6-1.

Table 10.6-1 Processor ERRa Command Summary

Command Name	Function	Default
SET CONSTRAINT_SET	Specifies constraint-set number for error accumulation	1
SET LDI	Specifies logical device index of computational database	1
SET LOAD_SET	Specifies load-set number for error accumulation	1
SET MESH	Specifies mesh number for error accumulation	0
SET STEP	Specifies load/time-step number for error accumulation	0
ACCUMULATE	Accumulate error estimates; store in database	

10.6.3 Command Definitions

10.6.3.1 ACCUMULATE Command

This is the "go" command for processor ERRa. It causes ERRa to accumulate element errors for all elements in a specified mesh, and to output them to the element error table (EET) dataset *EltNam*.ERROR.*ldset.conset.mesh* (*EltNam* is the element name, *ldset* is the load set number, *conset* is the constraint set number, and *mesh* is the mesh number).

Command syntax:

ACCUMULATE

10.6.3.2 SET CONSTRAINT_SET Command

This command defines the constraint set number associated with the element solution data for which error estimates are to be computed. This number should appear as the second cycle number in names of all element solution datasets (e.g., STRESS, STRAIN, and STRAIN_ENERGY). Relevant only for linear static analysis.

Command syntax:

SET CONSTRAINT_SET = *constraint_set*

where

Parameter	Description
constraint_set	Constraint set number (default value: 1)

10.6.3.3 SET LDI Command

This command defines the logical device index for the central database.

Command syntax:

SET LDI = *ldi*

where

Parameter	Description
ldi	Logical device index (default value: 1)

10.6.3.4 SET LOAD_SET Command

This command defines the load set number associated with the element solution data for which error estimates are to be computed. This number should appear as the first cycle number in names of all element solution datasets (e.g., STRESS, STRAIN, and STRAIN_ENERGY). Relevant only for linear static analysis.

Command syntax:

SET LOAD_SET = *load_set*

where

Parameter	Description
load_set	Load set number (default value: 1)

10.6.3.5 SET MESH Command

This command defines the mesh number associated with the model and solution data for which error estimates are to be computed. This number should appear as the third cycle number in names of all datasets (e.g., *EltNam*.ERROR.*ldset.conset.mesh*).

Command syntax:

SET MESH = *mesh*

where

Parameter	Description
mesh	Mesh number to be processed (default value: 0)

10.6.3.6 SET STEP Command

This command defines the solution step number associated with the element solution data for which error estimates are to be computed. This number should appear as the first cycle number in names of all element solution datasets (e.g., STRESS, STRAIN, and STRAIN_ENERGY). Relevant only for linear static analysis.

Command syntax:

SET STEP = *step*

where

Parameter	Description
step	Solution step number (default value: 0—implies linear analysis)

10.6.4 Database Input/Output

10.6.4.1 Input Datasets

A summary of input datasets required by Processor ERRa is given in Table 10.6-2.

Table 10.6-2 Processor ERRa Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset
EltNam.DEFINITION...*mesh*	EDT	Element definition datasets
EltNam.ERROR.*ldset.conset.mesh*	EET	Element error datasets. The following element attributes are created

10.6.4.2 Output Datasets

A summary of output datasets/attributes created by Processor ERRa is given in Table 10.6-3.

Table 10.6-3 Processor ERRa Output Datasets

Dataset/Attribute	Class	Contents	
EltNam.ERROR.*ldset.conset.mesh*	EET	Element error datasets. The following element attributes are created.	
		Attribute	**Description**
		AbsErr	Absolute element error E_e
		Energy	Element strain-energy density: $U_e = \int_{\Omega_e} \rho^{FE} d\Omega$ where $\rho^{FE} = \sigma:\varepsilon$
		EngGrd	Element strain-energy density gradient: $\nabla\rho = \frac{\rho_{max} - \rho_{min}}{\sqrt{A_e}}$ where A_e is the element area.
		ErrRat	Element error ratio: $\tilde{E}_e = \frac{E_e/A_e}{\max_e (E_e/A_e)}$
		RelErr	Relative element error: $\hat{E}_e = \frac{E_e}{\sqrt{U_{tot}^{FE}/Nel}}$ where U_{tot}^{FE} is the total finite element strain energy integrated over all elements, and Nel is the total number of elements.

10.6.5 Limitations

There are no serious limitations associated with this processor.

10.6.6 Error Messages

ERRa contains extensive error checking. Most of the error messages printed by ERRa are self-explanatory and aim to help the user correct mistakes. Some of the errors may occur at code levels below ERRa (e.g., HDB, DB, GAL, etc.), and ERRa describes those errors to the best of its ability. The following summarizes error messages related to user interface problems as produced by ERRa.

Index	Error Message	Cause	Recommended User Action
1	Unknown *set variable name* encountered	ERRa user interface cover encountered an unrecognized SET variable name.	Check spelling of *variable name* in CLIP procedure.
2	Unknown *command* encountered.	ERRa user interface cover encountered an unrecognized command.	Check spelling of *command* in CLIP procedure.
3	*Old/new dataset* name could not be opened.	ERRa could not open a certain dataset.	1. Check the execution log file; look for error produced by processors prior to ERRa execution. 2. Try to verify the particular dataset using the HDBprt processor. 3. Make sure that all input datasets are present in the database file.
4	*Dataset name* could not be closed.	ERRa could not close a certain dataset.	1. Check the execution log file; look for errors previously produced by processor ERRa. 2. Verify that ERRa is the only processor accessing the database file (is ARGx being used in the same directory?).
5	*Dataset name* access problem encountered.	ERRa could not get/put an attribute from the dataset name table.	Verify that the particular dataset contain attributes required by ERRa (e.g., EST contains nontrivial data at integration point).

10.6.7 Examples and Usage Guidelines

10.6.7.1 Example 1: Basic Operation

```
RUN ERRa

                SET MESH        = 1

                ACCUMULATE

STOP
```

In this example, all default options are chosen except for the mesh number.

10.6.8 References

None.

10.7 Processor ERRSM (Error Estimates: Smoothing-Based)

10.7.1 General Description

Processor ERRSM computes element error estimates employing smoothed stresses, strains, or strain energy densities (and possibly their gradients) which are assumed to have been computed in advance by a separate smoothing processor (See Chapter 9). The smoothed field is compared to the original finite element approximation of the field to yield an estimate of the element displacement error expressed in terms of the strain energy norm.

The definition of the absolute element error estimate E_e computed by ERRSM is as follows. For ERROR_MEASURE = STRAIN:

$$E_e = \frac{1}{2}\sqrt{\int_{\Omega_e} (\varepsilon^{SM} - \varepsilon^{FE})^T C (\varepsilon^{SM} - \varepsilon^{FE}) d\Omega}$$

or, for ERROR_MEASURE = STRESS:

$$E_e = \frac{1}{2}\sqrt{\int_{\Omega_e} (\sigma^{SM} - \sigma^{FE})^T C (\sigma^{SM} - \sigma^{FE}) d\Omega}$$

or, for ERROR_MEASURE = STRAIN_ENERGY:

$$E_e = \sqrt{\int_{\Omega_e} |\hat{U}^{SM} - \hat{U}^{FE}| d\Omega}$$

ERRSM currently works in conjunction with a standard smoothing processor, and its limitations depend largely on the limitations of the smoothing processor (e.g., processor SMT). The output of ERRSM consists of several attributes in the element error table (EET) data object (dataset name *EltNam*.ERROR.*), which may be used for adaptive refinement, or just as an assessment of element errors for a given mesh.

Processor ERRSM is normally invoked indirectly via procedure EST_ERR_SM, which is called automatically by the adaptive analysis control procedure AR_CONTROL.

10.7.2 Command Summary

See Section 10.2 for the summary of the generic commands common to all error estimation processors. Two additional commands are required by processor ERRSM, as described in Table 10.7-1.

Table 10.7-1 Special Processor ERRSM Commands

Command Name	Function	Default
SET SMOOTH_LOCATIONS	Specifies where smoothed element quantities are stored: at integration points or nodes.	INTEG_PTS
SET SMOOTH_GRADIENTS	Indicates whether smoothed gradients are to be used to compute error estimates (<true> or <false>).	<false>

10.7.3 Command Definitions

See Section 10.2 for the definition of the generic commands common to all error estimation processors.

10.7.3.1 SET SMOOTH_LOCATIONS Command

This command indicates where the smoothed element strains, stresses, or strain energy densities have been evaluated. Unsmoothed quantities are always assumed to be stored at element integration points.

Command syntax:

```
SET SMOOTH_LOCATIONS = { INTEG_PTS | NODES }
```

where

Parameter	Description
INTEG_PTS	Smoothed quantities are stored at element integration points
NODES	Smoothed quantities are stored at element nodes (and must be interpolated to integration points in order to compare with unsmoothed quantities.

10.7.3.2 SET SMOOTH_GRADIENTS Command

This command indicates whether or not smoothed gradients are to be employed in the error estimates.

Command syntax:

```
SET SMOOTH_GRADIENTS = { <true> | <false> }
```

where

Parameter	Description
<true>	Gradients of stress, strain, or strain energy density will be employed in the element error estimates.
<false>	Gradients will be ignored (default).

10.7.4 Database Input/Output

10.7.4.1 Input Datasets

A summary of input datasets required by Processor ERRSM is given below in Table 10.7-2.

Table 10.7-2 Processor ERRSM Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset
EltNam.INTERPOLATION...*mesh*	EIT	Element interpolation datasets
EltNam.DEFINITION...*mesh*	EDT	Element definition datasets
EltNam.FABRICATION...*mesh*	EFT	Element fabrication datasets
EltNam.STRAIN.*id1.id2.mesh;* or: *EltNam*.STRESS.*id1.id2.mesh;* or: *EltNam*.STRAIN_ENERGY.*id1.id2.mesh*	EST	Element strain, stress, or strain energy datasets, depending on the SET ERROR_MEASURE command. The values stored are expected to be element integration points in a globally meaningful coordinate frame.
EltNam.STRAIN_SM.*id1.id2.mesh;* or: *EltNam*.STRESSSM.*id1.id2.mesh;* or: *EltNam*.STRAIN_ENERGY_SM.*id1.id2.mesh*	EST	Smoothed element strain, stress, or strain energy datasets, depending on the error measure selected by the SET ERROR_MEASURE command. The values stored are expected to be either at element integration points or element nodes, depending on the SET SMOOTH_LOCATIONS command, in the same coordinate frame as the unsmoothed quantities.
NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate dataset

10.7.4.2 Output Datasets

A summary of datasets/attributes output by Processor ERRSM is given in Table 10.7-3.

Table 10.7-3 Processor ERRSM Output Datasets

Dataset	Class	Contents	
EltNam.ERROR.*id1.id2.mesh**	EET	Element error datasets. The following element attributes are created:	
		Attribute	**Description**
		AbsErr (E_e)	Absolute element error E_e. For ERROR_MEASURE=STRAIN: $\frac{1}{2}\sqrt{\int_{\Omega_e}(\varepsilon^{SM}-\varepsilon^{FE})^T C(\varepsilon^{SM}-\varepsilon^{FE})d\Omega}$ or, for ERROR_MEASURE=STRESS: $\frac{1}{2}\sqrt{\int_{\Omega_e}(\sigma^{SM}-\sigma^{FE})^T C(\sigma^{SM}-\sigma^{FE})d\Omega}$ or, for ERROR_MEASURE=STRAIN_ENERGY: $\sqrt{\int_{\Omega_e}\lvert\hat{U}^{SM}-\hat{U}^{FE}\rvert d\Omega}$
		Energy (U_e^{FE})	Element strain-energy: $U_e^{FE}=\int_{\Omega_e}(\hat{U})^{FE}d\Omega$ where $\hat{U}^{FE}=\sigma^t\varepsilon/2$.
		EngGrd (U_e^{SM})	Smoothed element strain-energy: $U_e^{SM}=\int_{\Omega_e}(\hat{U})^{SM}d\Omega$ where $\hat{U}^{SM}=(\varepsilon^{SM})^t C\varepsilon^{SM}/2$.

*—created dataset

10.7.5 Limitations

10.7.5.1 Partitioning Requirement

As with most smoothing-based error estimators, it is necessary to partition the elements so that physical discontinuities (e.g., thickness jumps, point forces, or intersections in built-up structures) occur only on the boundary of element partitions. Otherwise, excessive refinement in the vicinity of the physical discontinuity may result.

10.7.5.2 Common Strain Coordinate System

ERRSM subtracts and integrates element basic and smoothed strain, stress, and/or strain-energy density quantities evaluated at element integration points or nodes. For the strain and stress options (i.e., error measures) all strains or stresses must be expressed in a consistent coordinate system, which must be identical for smoothed and basic values. The user can assure this by choosing a meaningful stress direction option (STR_DIRECTION) when invoking the adaptive analysis control procedure.

10.7.6 Error Messages

ERRSM contains extensive checking. Most of the error messages printed by ERRSM are self-explanatory messages and aim to help the user correct mistakes. Some errors may occur at code levels below ERRSM (e.g., HDB, DB, GAL, etc.); ERRSM describes them to the best of its ability.

The following summarizes error messages related to user interface problems produced by ERRSM.

Index	Error Message	Cause	Recommended User Action
1	Unknown *set variable name* encountered	ERRSM user interface cover encountered an unrecognized SET variable name.	Check spelling of *variable name* in CLIP procedure.
2	Unknown *command* encountered.	ERRSM user interface cover encountered an unrecognized command.	Check spelling of *command* in CLIP procedure.
3	*Old/new dataset name* could not be opened.	ERRSM could not open a certain dataset.	1. Check the execution log file; look for error produced by processors prior to ERRSM execution. 2. Try to verify the dataset name using the HDBprt processor. 3. Verify all input datasets are in database file.
4	*Dataset name* could not be closed.	ERRSM could not close a certain Dataset.	1. Check the execution log file for errors previously produced by processor ERRSM. 2. Verify ERRSM is the only processor accessing the database file (is ARGx used in the same directory?).

Index	Error Message	Cause	Recommended User Action
5	*Dataset name* access problem encountered.	ERRSM could not get/put an attribute from dataset name table.	Verify dataset contains attributes required by ERRSM (e.g., EST contains nontrivial data at integration point).

10.7.7 Examples and Usage Guidelines

10.7.7.1 Example 1: Basic Operation

```
RUN ERRSM
              SET MESH              = 1
              ESTIMATE ERRORS
STOP
```

In this example, all default options are chosen except for the mesh number.

10.7.7.2 Example 2: Two Element Groups Partition

```
RUN ERRSM
              SET MESH              = 1
              SET ELEMENT_GROUP     = 1
              ESTIMATE ERRORS
              SET ELEMENT_GROUP     = 2
              ESTIMATE ERRORS
STOP
```

In this example, all default options are chosen except for the mesh number and element group. For each group, a separate "go" command is issued to ensure that no smoothing will take place along the boundaries between the first and second groups of elements.

10.7.8 References

[1] Zienkiewicz, O. C., and Zhu, J. Z., "A Simple Error Estimator For Adaptive Procedure for Practical Engineering Analysis," *International Journal of Numerical Engineering*, Vol. 24, pp. 337-357, 1987.

11 Mesh Refinement Processors

11.1 Overview

In this chapter, COMET-AR mesh refinement processors, typically used in the context of adaptive refinement (AR), are described. The convention is to call these processors REF*i*, and allow individual researchers to develop their own processors. Differences in REF*i* processor commands can be covered by writing tailor-made versions of the mesh refinement procedure REF_MESH_1 (see Section 5.8). Some conventions, and perhaps a template, have been established by processor REF1, which is the first AR-compatible mesh refinement processor to be developed for COMET-AR.

Table 11.1-1 Outline of Chapter 11: Mesh Refinement Processors

Section	Processor	Function
11.2	REF1	Mesh Refinement Processor; contains various forms of adaptive *h* and uniform *p* refinement schemes.

The command language and database requirements for the above adaptive mesh refinement processors conform to common conventions. This is to facilitate their use by high-level solution procedures such as AR_CONTROL in the context of adaptive refinement.

11.2 Processor REF1 (Mesh Refinement: $h_c/h_s/h_t/p$)

11.2.1 General Description

Processor REF1 performs one stage of adaptive mesh refinement (and/or unrefinement) based on previously computed element error estimates (e.g., generated by one of the ERR*i* processors) for a given mesh. Presently, the mesh refinement options implemented in REF1 include transition-based *h*-refinement (or h_t-refinement), cons traint-based *h*-refinement (or h_c-refinement), superposition-based *h*-refinement (or h_s-refinement) and/or uniform *p*-refinement employing Lagrange/ANS-type quadrilateral plate/shell elements. The expected database input for REF1 is a complete set of model definition datasets, plus an element error table (EET) dataset containing element error estimates for a particular mesh, *m*. As output, REF1 creates an entirely new set of model definition datasets for mesh *m*+1.

REF1 provides two solid-model interface (SMI) options: discrete and user-defined. With the discrete SMI option, REF1 views the initial finite element model as the exact model for geometry, materials, loads, and boundary conditions. With the user-defined SMI option, REF1 calls user-written subroutines to obtain this data at newly created nodes and element integration points (see Chapter 16, *Solid Model Interface*).

Processor REF1 is typically invoked by a high-level AR control procedure, such as AR_CONTROL (via procedure REF_MESH_1), in an adaptive refinement iteration loop.

11.2.2 REF1 — Refinement Techniques

The mesh refinement processor REF1 includes a variety of mesh refinement techniques including several mesh partition techniques (h_t-refinement) and polynomial enrichment technique (*p*-refinement). Each of the available refinement techniques will be briefly described in the following subsections.

11.2.2.1 Transition-Based Refinement Techniques—h_t, h_{tt}, h_{tq}-refinement

Transition-based *h*-refinement techniques employ special refinement patterns to transition from refined mesh zones to neighboring, unrefined zones. REF1 includes three methods of transitioning from refined to unrefined zones:

h_t ⇒ Transition zones employ quadrilateral-only patterns (for an all quadrilateral element mesh);

h_{tt} ⇒ Transition zones employ triangular-only patterns (for an all triangular element mesh);

h_{tq} ⇒ Transition zones employ mixed quadrilateral and collapsed quadrilateral triangular elements patterns.

These three transition techniques and the patterns they produce are shown in Figure 11.2-1.

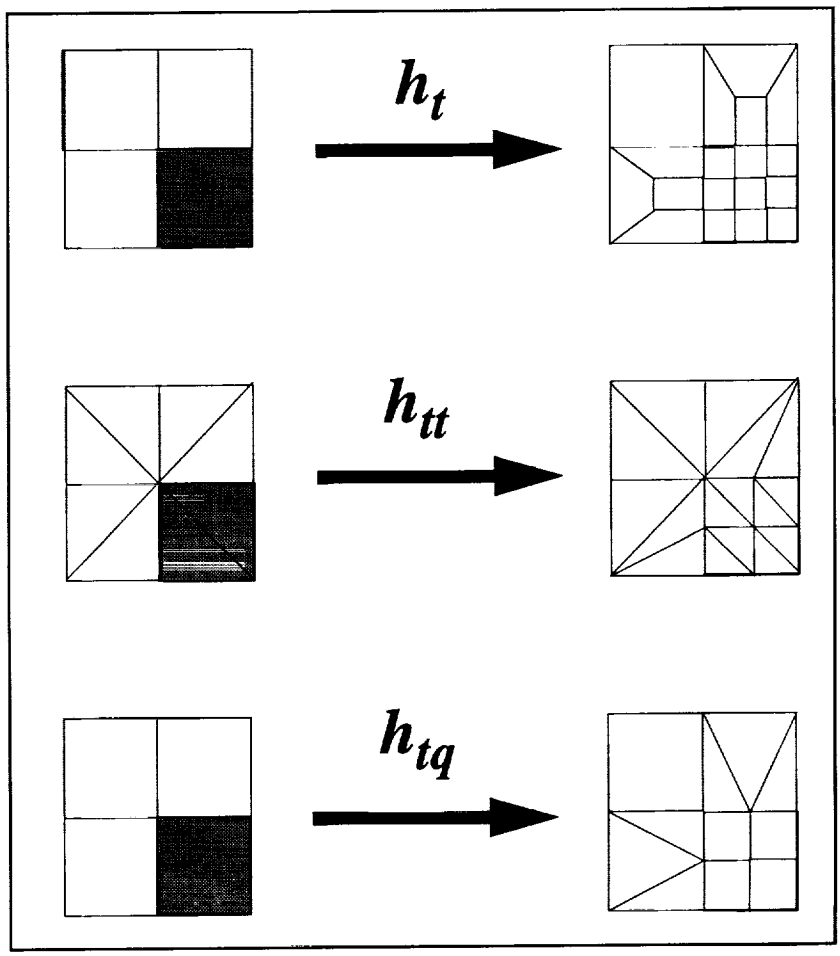


Figure 11.2-1 Transition-Based Refinement Techniques

11.2.2.2 Constraint-Based Refinement Technique—h_c-refinement

Constraint-based *h*-refinement techniques employ a special displacement field constrained to ensure compatibility (or continuity) of the displacement field across boundaries between refined mesh zones and neighboring, unrefined zones.

REF1 employs standard Lagrange constraints for enforcing the compatibility of the displacement field as shown in Figure 11.2-2.

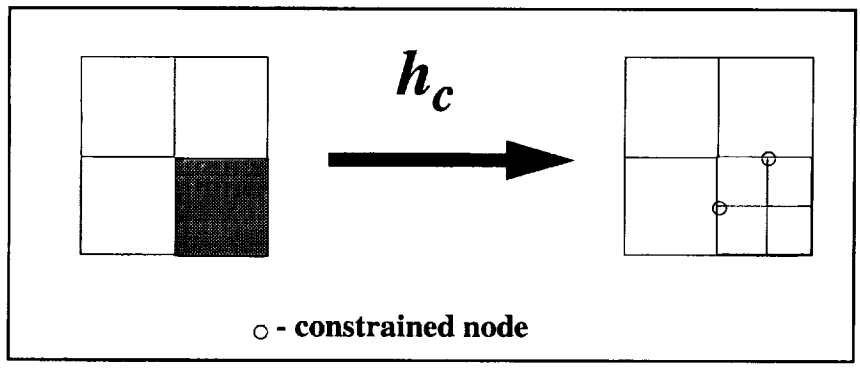


Figure 11.2-2 Constraint-Based Refinement Technique

11.2.2.3 Superposition-Based Refinement Technique—h_s-refinement

Superposition-based *h*-refinement techniques add a second refined mesh on top of an existing mesh. New degrees of freedom associated with new nodes in the superimposed mesh are treated as relative (or incremental) degrees of freedom. Compatibility is maintained in this method by simply suppressing the relative displacements along the interface boundaries between the underlying mesh and the superposed mesh as shown in Figure 11.2- 3.

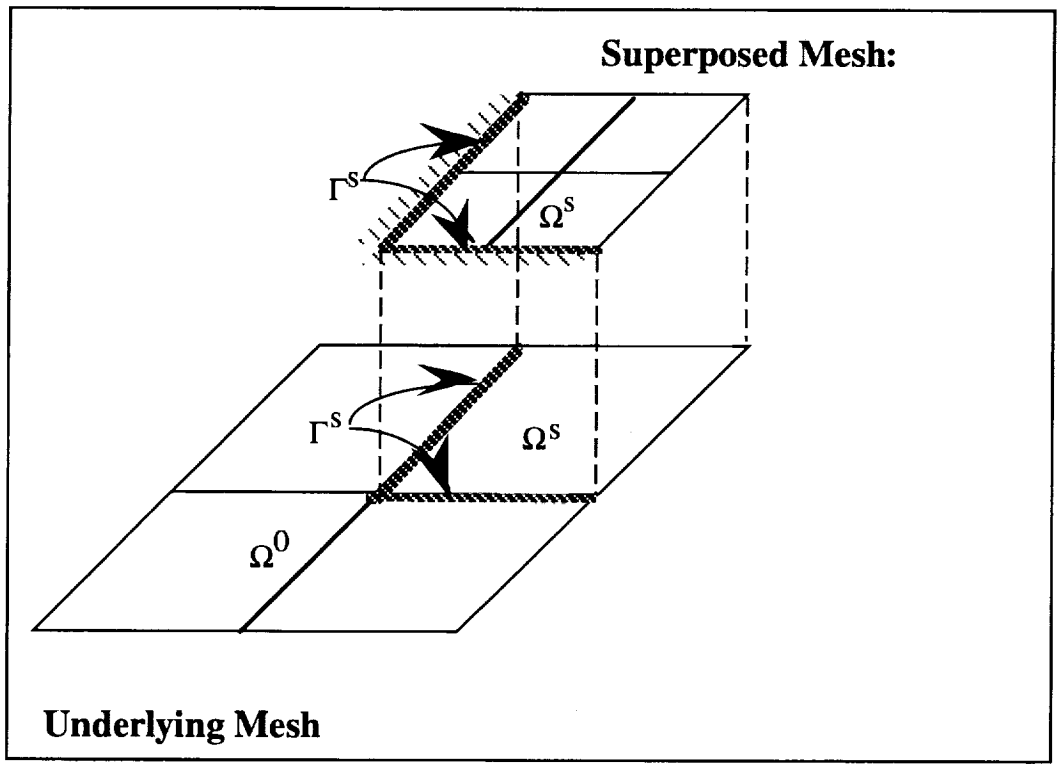


Figure 11.2-3 Superposition-Based Refinement Technique

11.2.2.4 Uniform Polynomial Enrichment Refinement Technique—p_u-refinement

Uniform polynomial enrichment p_u-refinement increases the polynomial order of all elements in the mesh as shown in Figure 11.2-4. This refinement option is only applicable in conjunction with the variable order element processors ES*i*p.

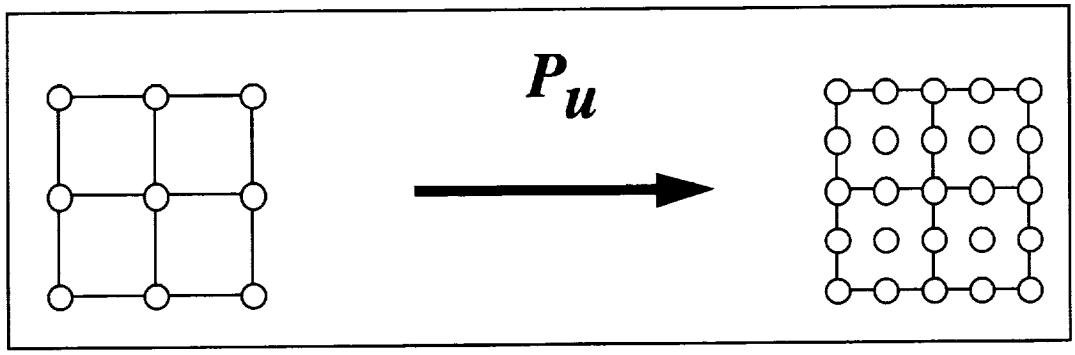


Figure 11.2-4 Uniform Polynomial Enrichment Refinement Technique

11.2.3 REF1—Multi-Level and Multi-Technique Refinement Control

The mesh refinement processor REF1 is capable of multi-level refinement and unrefinement (within a single AR iteration) and includes a preliminary implementation of multi-technique refinement (e.g., using both *h*-refinement and *p*-refinement in a single iteration).

User control of multi-level refinement is illustrated in Figure 11.2-5. The user can specify *Num_Ref_Tols* tolerance values and corresponding *Ref_Levels* for controlling the refinement based on the element *Refine_Indicator* being used (e.g., MAX or AVE options). If REF1 encounters an element with an error measure in the range $[Ref_Tols_i, Ref_Tols_{i+1}]$ then Ref_Levels_i levels of refinements will be used for that element. User control for unrefinement is similar to the refinement control described above.

Effective use of this refinement control algorithm requires the use of more refinement levels for elements with high errors than for elements with low errors, and similarly elements with very low errors should be allowed to unrefine more levels than elements with moderately low errors.

In addition to the multi-level refinement control, REF1 provides the user with an option to mix *h*- and *p*-refinement within a single refinement iteration loop. Figure 11.2-6 illustrates the control arguments used for this purpose. The user can specify control points on the element energy gradient axis: the *p_gradient* and the *h_gradient*.

The idea here is to take advantage of the special characteristics of *h*- and *p*-refinement. *p*-refinement is extremely effective in capturing monotonic changes in the solution field and the algorithm employs pure *p*-refinement in the low range of element energy gradients. Rapid changes in the solution field are more adequately captured by the *h*-refinement method and pure

h-refinement is employed in the mid-range values. Finally, for the upper range of element energy gradients, both methods are employed simultaneously for refinement.

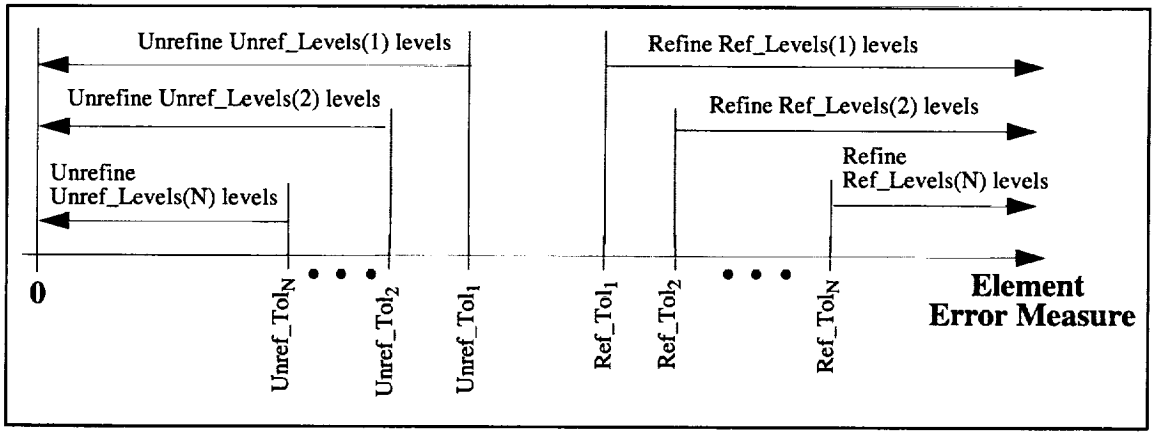


Figure 11.2-5 Multi-Level Refinement—User Control

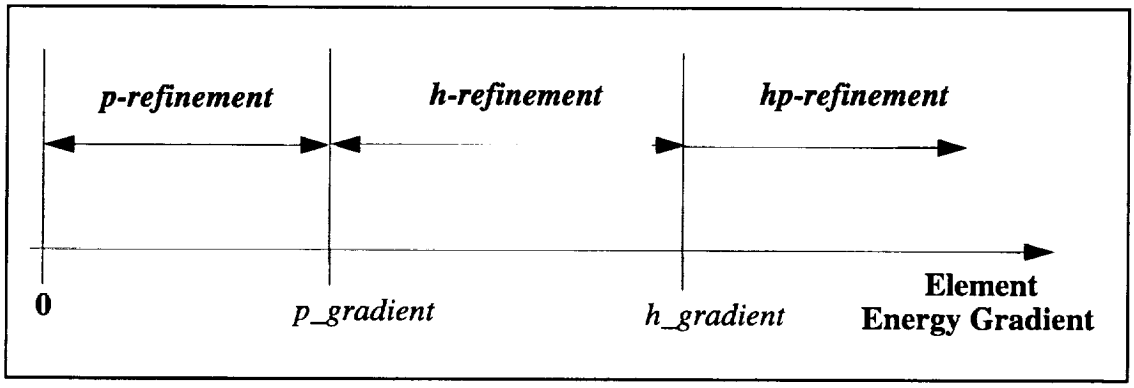


Figure 11.2-6 Multiple Methods Refinement—User Control

A general *p-refinement* technique is not yet implemented in REF1 (only the uniform pu-refinement capability is implemented). DO NOT USE THE MULTI-METHOD CONTROL OPTION YET!

11.2.4 Command Summary

Processors REF1 follows standard COMET-AR command interface protocol. A summary of REF1 commands is given in Table 11.2-1.

Table 11.2-1 Processor REF1 Command Summary

Command Name	Function	Default Value
SET CONSTRAINT_SET	Specifies constraint-set number	1
SET H_GRADIENT	Relative energy gradient mark above which both *h* and *p*-refinement will occur (for mixed *h*/*p*-refinement options)	0.0
SET LDI	Specifies logical device index of computational database	1
SET LOAD_SET	Specifies load-set number	1
SET MAX_ASPECT_RATIO	Distortion control parameters	0.0,0.0
SET MAX_h_LEVEL	Maximum number of *h*-refinement levels allowed	10
SET MAX_p_LEVEL	Maximum number of *p*-refinement levels allowed	5
SET MESH/NEW	Specifies new (generated) mesh number for refinement	1
SET MESH/OLD	Specifies old (reference) mesh number for refinement	0
SET NUM_REFINE_TOLS	Number of refinement tolerances	2
SET NUM_UNREFINE_TOLS	Number of unrefinement tolerances	0
SET P_GRADIENT	Relative energy gradient mark below which only p-refinement will occur (for mixed h/p-refinement options)	0.0
SET REFINE_DIRS	Allowable refinement directions	1,2,3
SET REFINE_LEVELS	Number of refinement levels for each refinement tolerance	1,2
SET REFINEMENT_INDICATOR	Specifies error quantity to be used for setting refinement indicators option	MAX
SET REFINEMENT_TECHNIQUE	Specifies refinement estimation option	h_t
SET REFINE_TOLS	Specifies refinement tolerances	0.90,0.95
SET STEP	Specifies load/time-step number	0
SET UNREFINE_LEVELS	Number of unrefinement levels for each unrefinement tolerance	0
SET UNREFINE_TOLS	Specifies unrefinement tolerances	0.0
REFINE_MESH	Refine the reference mesh	

11.2.5 Command Definitions

11.2.5.1 REFINE_MESH Command

This is the "go" command for processor REF1. It causes REF1 to set the element's refinement indicators based on element errors previously computed by an ERR*i* processor, and to adaptively refine the reference mesh, *m*, and generate a complete database for the next, adaptively refined mesh, $m + 1$.

Command syntax:

REFINE_MESH

11.2.5.2 SET CONSTRAINT_SET Command

This command defines the constraint set number associated with the element and nodal data in both the reference and the refined meshes. This number should appear as the second cycle number in names of all element and nodal datasets.

Command syntax:

SET CONSTRAINT_SET = *conset*

where

Parameter	Description
conset	Constraint set number (default value: 1)

11.2.5.3 SET H_GRADIENT Command

This command defines the h_gradient mark on the element energy gradient axis for multi-technique refinement (see Section 11.2.3, *Multi-Level and Multi-Technique Refinement Control*).

Command syntax:

SET H_GRADIENT = *h_gradient*

where

Parameter	Description
h_gradient	h_gradient mark value (default value: 0.0)

11.2.5.4 SET LDI Command

This command defines the logical device index for the computational database.

Command syntax:

SET LDI = *ldi*

where

Parameter	Description
ldi	Logical device index (default value: 1)

11.2.5.5 SET LOAD_SET Command

This command defines the load set number associated with the element data in both the reference and the refined meshes. This number should appear as the first cycle number in names of all element load datasets.

Command syntax:

SET LOAD_SET = *ldset*

where

Parameter	Description
ldset	Load set number (default value: 1)

11.2.5.6 SET MAX_ASPECT_RATIO Command

This command defines a distortion control parameters for h_t-refinement. This option allows the user to maintain some distortion control of the refined mesh. REF1 is capable of checking two levels of aspect-ratio measures.

- ***Parent.*** This is a pre-refinement check. Set this distortion control value to force an element to refine uniformly if the element's aspect-ratio is greater than this value. (The transition refinement patterns always increases the aspect-ratio in the generated elements while uniform refinement maintains the parent element aspect-ratio.)
- ***Child.*** This is a post-refinement check. Set this distortion control value to force an element to refine uniformly if any of its child element's aspect-ratio is greater than this value.

Command syntax:

SET MAX_ASPECT_RATIO = *parent, child*

where

Parameter	Description
parent	Pre-refinement max aspect-ratio for non-uniform refinement (default value: 0.0—no distortion control for the parent element)
child	Post-refinement max aspect-ratio for non-uniform refinement (default value: 0.0—no distortion control for the child element)

11.2.5.7 SET MAX_H_LEVEL Command

This command defines the maximum allowable *h*-refinement level. REF1 will not allow any element in the original mesh to refine more than MAX_H_LEVEL levels (i.e., no more than MAX_H_LEVEL generations of an element may exist in the refined mesh).

Command syntax:

SET MAX_H_LEVEL = *max_h_level*

where

Parameter	Description
max_h_level	Maximum level of refinement in the refined mesh. (default value: 10)

11.2.5.8 SET MAX_P_LEVEL Command

This command defines the maximum allowable *p*-refinement level. REF1 will not allow any element in the original mesh to have shape function polynomials of order higher than MAX_P_LEVEL order.

Command syntax:

SET MAX_P_LEVEL = *max_p_level*

where

Parameter	Description
max_p_level	Maximum element polynomial order in the refined mesh (default value: 5)

11.2.5.9 SET MESH/NEW Command

This command defines the mesh number associated with the refined model data. REF1 will use this number as the third cycle number in names of all datasets associated with the refined mesh.

Command syntax:

SET MESH/NEW = *new_mesh*

where

Parameter	Description
new_mesh	Refined mesh number (default value: *old_mesh* + 1)

11.2.5.10 SET MESH/OLD Command

This command defines the mesh number associated with the reference model and solution data. REF1 will use this mesh as a reference mesh and will adaptively refine this mesh.

Command syntax:

SET MESH/OLD = *old_mesh*

where

Parameter	Description
old_mesh	Reference mesh to be refined (default value: 0)

11.2.5.11 SET NUM_REFINE_TOLS Command

This command defines the number of refinement tolerances to be used by REF1 in setting the refinement indicators (see Section 11.2.3, *Multi-Level and Multi-Technique Refinement Control).*

Command syntax:

SET NUM_REFINE_TOLS = *num_tols*

where

Parameter	Description
num_tols	Number of tolerances used for controlling the refinement (default value: 2)

11.2.5.12 SET NUM_UNREFINE_TOLS Command

This command defines the number of unrefinement tolerances to be used by REF1 in setting the unrefinement indicators (see Section 11.2.3, *Multi-Level and Multi-Technique Refinement Control*).

Command syntax:

SET NUM_UNREFINE_TOLS = *num_tols*

where

Parameter	Description
num_tols	Number of tolerances used for controlling the unrefinement (default value: 0)

11.2.5.13 SET P_GRADIENT Command

This command defines the p_gradient mark on the element energy gradient axis for multi-method refinement (see Section 11.2.3, *Multi-Level and Multi-Technique Refinement Control*).

Command syntax:

SET P_GRADIENT = *p_gradient*

where

Parameter	Description
p_gradient	p_gradient mark value (default value: 0.0)

11.2.5.14 SET REFINE_DIRS Command

This command defines the allowable refinement directions in the element frame directions. In certain cases the user may wish to use this option to restrict the refinement in a certain direction for a more efficient solution.

Command syntax:

SET REFINE_DIRS = Dir_1, Dir_2,...

where

Parameter	Description
Dir_i	The *i*th element direction flag (default value: 1,2,3)

11.2.5.15 SET REFINE_LEVELS Command

This command defines the number of refinement levels to be used by REF1 for each refinement tolerance mark (see Section 11.2.3, *Multi-Level and Multi-Technique Refinement Control*).

Command syntax:

SET REFINE_LEVELS = $Level_1$, $Level_2$,..., $Level_{Num_Ref_Tols}$

where

Parameter	Description
$Level_i$	Number of refinement levels to be used for refining an element whose error measure is in the range $[Tol_i, Tol_{i+1}]$ (default value: 1, 2)

11.2.5.16 SET REFINE_INDICATOR Command

This command defines the error measure indicator to be used for setting the element refinement indicators.

Command syntax:

SET REFINE_INDICATOR = *indicator*

where

Parameter	Description
indicator	Refinement indicator (default value: MAX)

Values for the *refine_indicator* parameter are listed below:

Refinement Indicator	Description
MAX	Use the element absolute error scaled by the maximum element error as the error measure for setting the element refinement indicators.
AVE	Use the element absolute error scaled by the square root of the average element strain energy as the error measure for setting the element refinement indicators.

11.2.5.17 SET REFINE_TECHNIQUE Command

This command defines the refinement technique for adaptively refining the reference mesh.

Command syntax:

SET REFINE_TECHNIQUE = *refinement_technique*

where

Parameter	Description
refine_technique	Refinement technique (default value: h_t)

Values for the *refine_technique* parameter are listed below:

Refinement Technique	Description
ht	h_t–refinement—transition-based refinement using quadrilateral only refinement patterns
htpu	h_t–refinement—transition-based refinement using quadrilateral only refinement patterns followed by p_u-refinement.
htt	h_{tt}–refinement—transition-based refinement using triangular only refinement patterns
htq	h_{tq}–refinement—transition-based refinement using mixed quadrilateral/triangular refinement patterns
htqpu	h_{tq}–refinement—transition-based refinement using mixed quadrilateral/triangular refinement patterns followed by p_u-refinement
pu	p_u–refinement—uniform polynomial enrichment refinement
hc	h_c–refinement—constraint-based refinement
hc3D	Three dimensional h_c–refinement—constraint-based refinement applicable only in conjunction with the 3D continuum-based shell element processors (ES36& ES37)
hs	h_s–refinement—superposition-based refinement

11.2.5.18 SET REFINE_TOLS Command

This command defines the values of the refinement tolerances to be used by REF1 in setting the refinement indicators (see Section 11.2.3, *Multi-Level and Multi-Technique Refinement Control*).

Command syntax:

SET REFINE_TOLS = Tol_1, Tol_2,..., $Tol_{Num_Ref_Tols}$

where

Parameter	Description
Tol_i	The *i*th tolerance value (default value: 0.90, 0.95)

11.2.5.19 SET STEP Command

This command defines the solution step number associated with the element and nodal data in both the reference and the refined meshes. This number should appear as the second cycle number in names of all element and nodal datasets. Relevant only for nonlinear analysis.

Command syntax:

SET STEP = *step*

where

Parameter	Description
step	Solution step number. (default value: 0)

11.2.5.20 SET UNREFINE_LEVELS Command

This command defines the number of unrefinement levels to be used by REF1 for each unrefinement tolerance mark (see Section 11.2.3, *Multi-Level and Multi-Technique Refinement Control*).

Command syntax:

SET UNREFINE_LEVELS = $Level_1$, $Level_2$,..., $Level_{Num_Unref_Tols}$

where

Parameter	Description
$Level_i$	Number of unrefinement levels to be used for unrefining an element whose error measure is in the range $[Tol_{i+1}, Tol_i]$ (default value: 0)

11.2.5.21 SET UNREFINE_TOLS Command

This command defines the values of the refinement tolerances to be used by REF1 in setting the unrefinement indicators (see Section 11.2.3, *Multi-Level and Multi-Technique Refinement Control*).

Command syntax:

SET UNREFINE_TOLS = Tol_1, Tol_2,..., $Tol_{Num_Unref_Tols}$

where

Parameter	Description
Tol_i	The *i*th unrefinement tolerance value (default value: 0.00)

11.2.6 Database Input/Output

11.2.6.1 Input Datasets

A summary of input datasets required by Processor REF1 is given in Table 11.2-2.

Table 11.2-2 Processor REF1 Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*old_mesh*	CSM	Model summary dataset
NODAL.COORDINATE...*old_mesh*	NCT	Nodal coordinate dataset
NODAL.DOF..*conset.old_mesh*	NDT	Nodal DOF dataset
NODAL.TRANSFORMATION...*old_mesh*	NTT	Nodal transformation dataset
NODAL.SPEC_FORCE.*ldset..old_mesh*	NVT	Nodal specified force dataset
NODAL.SPEC_DISP.*ldset..old_mesh*	NVT	Nodal specified displacement dataset
EltNam.DEFINITION...*old_mesh*	EDT	Element definition dataset
EltNam.REFINEMENT...*old_mesh*	ERT	Element refinement dataset (this dataset is created by REF1 for the initial mesh)
EltNam.INTERPOLATION...*old_mesh*	EIT	Element interpolation dataset
EltNam.ERROR.*ldset.conset.old_mesh*	EET	Element error dataset
EltNam.GEOMETRY...*old_mesh*	EGT	Element geometry (solid model links) dataset
EltNam.FABRICATION...*old_mesh*	EFT	Element fabrication dataset
EltNam.LOAD...*old_mesh*	ELT	Element loads datasets
LINE.REFINEMENT...*old_mesh*	LRT	Line refinement dataset (this dataset is created by REF1 for the initial mesh)
SURFACE.REFINEMENT...*old_mesh*	SRT	Surface refinement dataset (only in 3D-refinement—this dataset is created by REF1 for the initial mesh)

11.2.6.2 Output Datasets

A summary of output datasets created by Processor REF1 is given in Table 11.2-3.

Table 11.2-3 Processor REF1 Output Datasets

Dataset	Class	Contents
CSM.SUMMARY...*new_mesh**	CSM	Model summary dataset
NODAL.COORDINATE...*new_mesh**	NCT	Nodal coordinate dataset
NODAL.DOF..*conset.new_mesh**	NDT	Nodal DOF dataset
NODAL.TRANSFORMATION...*new_mesh**	NTT	Nodal transformation dataset
NODAL.SPEC_FORCE.*ldset..new_mesh**	NVT	Nodal specified force dataset
NODAL.SPEC_DISP.*ldset..new_mesh*	NVT	Nodal specified displacement dataset
EltNam.DEFINITION...*new_mesh**	EDT	Element definition dataset
EltNam.REFINEMENT...*new_mesh**	ERT	Element refinement dataset

Table 11.2-3 Processor REF1 Output Datasets (Continued)

Dataset	Class	Contents
EltNam.INTERPOLATION...*new_mesh**	EIT	Element interpolation dataset
EltNam.GEOMETRY...*new_mesh**	EGT	Element geometry (solid model links) dataset
EltNam.FABRICATION...*new_mesh**	EFT	Element fabrication dataset
EltNam.LOAD...*new_mesh**	ELT	Element loads datasets
LINE.REFINEMENT...*new_mesh**	LRT	Line refinement dataset
SURFACE.REFINEMENT...*new_mesh**	SRT	Surface refinement dataset (only in 3D refinement)

*—created dataset

11.2.7 Limitations

11.2.7.1 Distortion Sensitivity

Transition-based refinement (h_t-refinement) tends to generate distorted elements within the transition zones between refined and coarse mesh areas. Some of the shell elements (such as the ANS family) were found to be extremely sensitive to distortion and may cause a mesh locking phenomena in transition zones. Distortion control should be enforced when using such elements by judicious use of the SET MAX_ASPECT_RATIO command. This may alleviate some of the problem by reducing the amount of distortion in the refined mesh.

11.2.7.2 Consistent Constraints

Constraint-based refinement (h_c-refinement) requires a consistent set of displacement constraints for proper enforcement of the compatibility condition across element boundaries. For hybrid shell elements, such as the ANS family of elements, these constraints are not known and the automatic constraint builder algorithm employed by REF1 will substitute simple Lagrange constraints instead. This simple constraint equations may cause over constraining of nodes and may cause some spurious local errors in the vicinity of such nodes. In practice it was found that higher order hybrid elements, such as the 16ANS, were less sensitive to this type of approximate constraints then their lower order counterparts.

11.2.7.3 *p*-refinement Limitations

REF1 includes only uniform *p*-refinement capabilities which are restricted for use only in conjunction with the variable *p*-order element processors (e.g., ES1p and ES7p). The algorithm for setting refinement indicators includes provisions for general *p*-refinement and even mixed *hp*-refinement (see the SET H_GRADIENT and SET P_GRADIENT commands). Ignore these capabilities in the current version of REF1 and do not attempt to use either the general *p*-refinement or the mixed *hp*-refinement options.

11.2.7.4 3D Refinement Limitations

The capability for 3D-refinement in REF1 is limited to constraint-based (h_c)-refinement, and may be used only in conjunction with 3D elements based on variable-order, Lagrange-type brick element topologies (i.e., elements that have IxJxK nodal patterns, where I, J, and K are the number of nodes in each of the three natural coordinate directions).

11.2.8 Error Messages

REF1 contains extensive error checking. Most of the error messages printed by REF1 are self-explanatory and aim to help correct mistakes. Some of the errors may occur at code levels below REF1 (e.g., HDB, DB, GAL, etc.) and REF1 describes those errors to the best of its ability.

The following summarizes the error messages related to user interface problems as produced by REF1.

Index	Error Message	Cause	Recommended User Action
1	Unknown SET *variable name* encountered in REF1.	REF1 user interface cover encountered an unrecognized SET variable name.	Check spelling of *variable name* in CLIP procedure.
2	Unknown *command* encountered in REF1.	REF1 user interface cover encountered an unrecognized command.	Check spelling of *command* in CLIP procedure.
3	*Old/new dataset name* could not be opened in *routine name*.	REF1 could not open a certain dataset.	1. Check the execution log file; look for error produced by processors prior to REF1 execution. 2. Try to verify the particular *dataset name* using the HDBprt processor. 3. Make sure that all required input datasets are present in the database file.
4	*Dataset name* could not be closed in *routine name*.	REF1 could not close a certain dataset.	1. Check the execution log file; look for errors previously produced by processor REF1. 2. Verify that REF1 is the only processor accessing the database file (is ARGx being used in the same directory?).
5	*Dataset name* access problem encountered in *routine name*, —or— Could not get/put/add/update *attribute name* to *dataset name* in *routine name*.	REF1 could not get/put an attribute from/to the dataset name table.	Verify that the particular dataset contain attributes required by REF1 (e.g., EST contain nontrivial data at integration point).

Index	Error Message	Cause	Recommended User Action
6	Unknown Geometry Entity ID encountered in SMShl*xx*, *entity type* ID = *entityID*.	The solid model interface shell routines in REF1 could not locate a solid model geometry entity.	1. Verify that the elements are properly linked to the user written solid model definition routines (e.g., check Line_IDs and Surface_ID for the definition of the element in question. 2. Make sure that the version of REF1 being used is linked with the proper user written solid element model routines.
7	Solid Model Interface Problem encountered in SMShl*xx*.	The solid model interface shell routines in REF1 could not perform their current task.	1. Verify that the elements are properly linked to the user written solid model definition routines (e.g., check Line_IDs and Surface_ID for the definition of the element in question. 2. Make sure that the version of REF1 being used is linked with the proper user written solid element model routines.
8	Convergence problem encountered in *xxx* Proj — could not locate projected point along *geometry entity type*.	The solid model interface shell routines in REF1 could not project a new point into the boundaries of the corresponding solid model geometry entity.	1. Verify that the elements are properly linked to the user written solid model definition routines (e.g., check Line_IDs and Surface_ID for the definition of the element in question. 2. Make sure that the version of REF1 being used is linked with the proper user written solid element model routines. 3. Verify that the assumed parametric presentation for each type of geometry entity is maintained in the user written routine (i.e., each generic parameter varies in the bi-unit interval range, [-1,+1], and that a "one-to-one" mapping exist between the generic parametric space and the physical space for each geometry entity).

In addition to the above generic messages, REF1 will print any relevant information regarding the problem such as element data, nodal data, and geometry information to assist in correcting the error. A full trace-back printout of error messages will follow the first message, and REF1 will attempt to terminate its execution as cleanly as possible (closing opened datasets, releasing memory allocations, etc.).

11.2.9 Examples and Usage Guidelines

11.2.9.1 Example 1: Basic Operation

```
RUN REF1
              SET REFINE_TECHNIQUE      = hc
              SET REFINE_INDICATOR      = AVE
              SET NUM_REFINE_TOLS       = 1
              SET REFINE_TOLS           = 0.05
              SET REFINE_LEVELS         = 1
              SET MESH/OLD              = 0
              REFINE_MESH
STOP
```

In this example, reference mesh 0 is being refined (the refined mesh will be mesh 1) by up to one level of refinement using constraint-based refinement technique (h_c-refinement). All elements for which the relative element error is greater than 5% will be refined by dividing them into four elements.

11.2.9.2 Example 2: Multi-Level Refinement

```
RUN REF1
              SET REFINE_TECHNIQUE      = ht
              SET REFINE_INDICATOR      = MAX
              SET NUM_REFINE_TOLS       = 2
              SET REFINE_TOLS           = 0.90,  0.95
              SET REFINE_LEVELS         = 1, 2
              SET NUM_UNREFINE_TOLS     = 1
              SET UNREFINE_TOLS         = 0.10
              SET UNREFINE_LEVELS       = 1
              SET MESH/OLD              = 1
              REFINE_MESH
STOP
```

In this example, reference mesh 1 is being refined (the refined mesh will be mesh 2) by up to two levels of refinement using transition-based refinement technique (h_t-refinement). All elements for which the element max error ratio is greater than 90% will be refined by a single refinement level and elements having max error ratio greater than 95% will be refined by two refinement levels. Finally, all elements having a maximum error ratio less than 10% will be unrefined by one level.

11.2.10 References

[1] Stanley, G., Levit, I., Hurlbut, B., and Stehlin, B., *Adaptive Refinement Strategies for Shell Structures: Part 1: Preliminary Research*, 1991.

[2] Stanley, G., Levit, I., Hurlbut, B., Stehlin, B., Loden, W., and Swenson, L., *COMET–AR: Adaptive Refinement (AR) Manual*, May 1991.

12 Matrix/Vector Processors

12.1 Overview

In this chapter, COMET-AR matrix and vector algebra processors are described. These processors are typically invoked automatically by COMET-AR utility procedures, which in turn are invoked by high-level solution procedures. The matrix/vector processors currently available in COMET-AR and their section numbers within this chapter are given in Table 12.1-1.

Table 12.1-1 Outline of Chapter 12: Matrix/Vector Processors

Section	Processor	Function
12.2	ASM	Matrix assembly processor for SKYLINE and COMPACT matrix formats; also enforces multipoint constraints by direct elimination of dependent DOFs.
12.3	ASMs	Special matrix assembly processor required in conjunction with superposition type (h_s) mesh refinement.
12.4	ITER	Iterative linear equation solver based on pre-conditioned conjugate gradient method.
12.5	PVSOLV	Direct linear equation solver based on COMPACT matrix format; optimized for vector computers.
12.6	SKY	Direct linear equation solver based on SKYLINE matrix format; restricted to problems that fit in core.
12.7	SKYs	Special direct/iteration equation solver based on SKYLINE matrix format; needed with h_s mesh refinement.
12.8	VEC	General-purpose vector/pseudovector algebra utility.
12.9	VSS	Direct linear equation solver.

12.2 Processor ASM

12.2.1 General Description

The ASM assembly processor described here is an improved version of the initial (prototype) ASM processor described in Reference [1]. ASM was developed because previously existing matrix and vector assembly tools could not perform many of the operations required within the COMET-AR framework and it was frequently difficult, and sometimes impossible, to use those tools to conduct increasingly complex analyses and to treat problems in domains other than those for which they were originally developed. Additional motivation for developing ASM arose from anticipated needs for efficient treatment of structural analysis problems with transition regions (where many elements with nodes admitting all possible motions join other elements with nodes that only admit a subset of those freedoms), with nodes that may have more freedoms than the current processors allow, where different nodes have different numbers of degrees of freedom (DOF), and where different nodes have different types of DOF. Still more motivation for developing ASM stemmed from requirements for a more flexible tool to be used within the current and developing COMET-AR frameworks as new needs arise, including the need to assemble system matrices and associated right-hand-side (RHS) vectors using contributions stored in forms other than the venerable "EFIL" data structure that Testbed processors have historically used. The need to assemble these entities into data formats that current and future versions of COMET-AR processors recognize and treat, and the need for an assembly processor with the capability of adding new formats at both ends of the assembly process without the complications that usually accompany those efforts is evident.

Many applications require the imposition of multi-point constraints that occur naturally in the course of adaptive refinement, contact-impact, multibody-dynamic, and other analysis activities. Multi-point constraints, where some of the unknown (dependent) field variables are expressed in terms of other independent variables, are best treated by directly applying appropriate transformations to eliminate the dependent variables in favor of the independent variables at the element-level during the system matrix (and system vector) assembly process.

The direct elimination algorithm can be used to enforce explicit linear multi-point constraint relations, in which an element's degrees-of-freedom, $\boldsymbol{d^e}$, can be expressed in terms of a set of independent DOFs as follows:

$$d^e = \begin{Bmatrix} d_I^e \\ d_D^e \end{Bmatrix} = \begin{bmatrix} I & 0 \\ C_{DI}^e & C_{DI} \end{bmatrix} \begin{Bmatrix} d_I^e \\ d_I \end{Bmatrix} + \begin{Bmatrix} 0 \\ \alpha^e \end{Bmatrix}$$

or

$$d^e = C\bar{d} + \bar{\alpha}$$

Here an element's DOFs are logically partitioned into independent and dependent sets, $\boldsymbol{d^e_I}$ and $\boldsymbol{d^e_D}$, respectively. An element's dependent DOFs can generally be expressed in terms of the element's independent DOFs $\boldsymbol{d^e_I}$ (through coefficients $\boldsymbol{C^e_{DI}}$) and in terms of a second set of independent DOFs $\boldsymbol{d_I}$ (through coefficients $\boldsymbol{C_{DI}}$), where the $\boldsymbol{d_I}$ extends beyond the element's domain. For generality, an element's dependent DOFs are also influenced by a set of constant terms α^e.

Prior to constraint elimination, the local virtual work performed by an element's internal and external forces can be written as:

$$(\delta d^e)^T K^e d^e = (\delta d^e)^T f^e$$

where $\boldsymbol{K^e}$ is the element stiffness matrix and $\boldsymbol{f^e}$ is the element load vector. Using the previous expression for the element DOF vector $\boldsymbol{d^e}$, the constrained equilibrium equations can be written as:

$$\bar{K}\bar{d} = \bar{f}$$

where the constrained element stiffness matrix and load vector are given by:

$$\bar{K} = C^T K^e C$$

$$\bar{f} = C^T(f^e - K^e\bar{\alpha})$$

These are the terms assembled into the system stiffness matrix and system load vector.

The current version of processor ASM and some enhancements to it that are projected for the near future are described here. The current version of ASM is a second prototype which does many, but not yet all, of the necessary operations for solving new kinds of problems with the COMET-AR system. The functions of the current and projected versions of ASM are described in general terms in the remaining paragraphs of this section. The user interface to ASM (i.e., the commands that ASM recognizes and treats) is described in detail in the next section, with some discussions about the methodologies used by the processor.

12.2.1.1 Current and Projected Functionality

Basic GAL database operations (*OPEN, *CLOSE, *TOC, etc.) must be performed through CLIP directives [2]. ASM recognizes processor-specific commands that have been designed to make getting information into and out of the program as easy as possible.

ASM accepts information from the user-defined entities that contribute to the system matrix (and/or the system vector) to be assembled. These entities must be stored in datasets on one or more GAL library files. Currently, ASM can process contributions stored as any of the following data objects:

- An Element Matrix Table (EMT data object), used by COMET-AR processors for element contribution data. EMT contributions must be supplemented with additional information (nodal connectivity, etc.) stored in an Element Definition Table (EDT data object) and information in a Complete Model Summary Table (CSM data object). ASM uses HDB utilities [3] to access this information, shielding developer and user from data structure details;

- A Nodal Vector Table (NVT data object), used by COMET-AR processors for storage of nodally-oriented system vectors (for right-hand-side information);

- A System Vector Table (SVT data object), used by DOF-oriented processors for storage of computational system vectors which only contain information for the independent DOFs of the system.

ASM includes provisions for extension to treat system-matrix contributions stored in other self-descriptive formats in addition to (or instead of) those that are stored as EMT data objects:

- The SKY_MATRIX (skyline) format, used by the SKY processor (and other programs) for storage of a fully-assembled, symmetric system matrix;

- The COMPACT (compact-column) format, for space-efficient storage of an assembled unfactored, symmetric system matrix;

- The COMPAXX (compact-row) format, which also provides space-efficient storage of an assembled unfactored, symmetric system matrix and is compatible with new-generation solvers in use at NASA/LaRC and elsewhere.

ASM accepts information from the user that defines the important problem-size parameters (the highest node number and the maximum number of degrees of freedom at any node point, for example), that specifies the type of freedom for each potential DOF of the problem, the equation number assigned to each DOF that gets an equation number, and any constraints that need to be taken into account while or after assembling the system matrix and/or RHS vector. This information is stored in a Nodal DOF Table (NDT data object) and its related Complete Model Summary Table (CSM data object). These data structures, and others that ASM recognizes and treats, are described in Chapter 15, *Database Summary*.

ASM accepts information from the user defining the destination(s) and the format(s) for archival storage of the assembled system matrix and/or RHS vector. The following formats are available:

- COMPACT (assembled system matrix);
- COMPAXX (assembled system matrix);
- SKY_MATRIX (assembled system matrix);
- SVT data object (assembled computational vector).

ASM processes commands that request the immediate display of information about entities contributing to or defining the matrix (and/or RHS vector) being assembled, and that display information about the size and contents of the assembled matrix and/or vector.

ASM assembles a symmetric system matrix, summing all of the user-designated system-matrix contributions and taking all user-defined constraints into account, and stores the assembled system matrix on the user-specified output GAL library file(s) in the format(s) that the user requests.

ASM assembles right-hand-side computational vectors, summing the user-designated RHS contributions and taking user-defined constraints into account, and stores the assembled RHS vector on the user-specified output GAL library file(s).

ASM uses available topological information to facilitate and improve the efficiency of the assembly process. ASM currently uses the element- and nodal-level data in EMT-formatted contributions to partition its internal workspace and to define the order and nature of some of the assembly operations. When other contribution formats are introduced, ASM will be modified to perform some of its pre-assembly analyses at the DOF-level (when nodal-level information is not available).

12.2.2 Processor Command Summary

The user must employ CLIP directives to communicate directly with GAL database files and do general bookkeeping, branching, and arithmetic operations. CLIP directives are described in reference [2]. ASM-specific commands enable the user to access database-resident model-definition, DOF-table, element-contribution, RHS-contribution, and other information; to direct the flow of output from ASM to GAL-library storage locations; and to control certain aspects of ASM processor operations. ASM commands are summarized in Table 12.2-1 (shown in the order in which ASM commands would normally be entered).

Table 12.2-1 ASM Command Summary

Command Name	Function
MODEL	Specify the CSM (Complete Model Summary) table
RESET	Reset an assembly- or program-control parameter
INCLUDE	Define (or purge) contributions or constraints
OUTPUT	Define (or purge) output destination(s) and format(s)
SHOW	Display input, output, matrix, or vector information
ASSEMBLE	Assemble the system matrix and/or RHS vector
RESTART	Re-initialize ASM to treat a new assembly problem
STOP	Exit the ASM processor

12.2.3 Command Glossary

Complete descriptions of all of the current ASM commands are given in the following subsections. The commands are described in the same order as they are listed in Table 12.2-1.

12.2.3.1 MODEL Command

The first thing the ASM user must usually do is to specify the Complete Model Summary Table (CSM data object), which contains problem-size parameters and other vital information for the model to be treated. This is done with the MODEL command, the syntax of which is

MODEL [*ldi_csm* [*dsn_csm*]]

The MODEL command tells ASM to open the CSM data object on GAL library *ldi_csm* with name *dsn_csm*, and to extract two problem-size parameters: NNODES (the number of nodes for the model); and NDOFN (the maximum number of DOFs that may be associated with each node). The default values for *ldi_csm* and *dsn_csm* are 1 and CSM.SUMMARY. ASM also extracts from that CSM data object the mesh index (*mesh*) and any other information needed to perform operations requested via subsequent ASM commands.

The MODEL command is optional when the CSM data object is identified by the default values (*ldi_csm*=1 and *dsn_csm*=CSM.SUMMARY); it is required for any other situation.

The MODEL keyword may be abbreviated to one character.

12.2.3.2 RESET Command

The RESET command is used to reset an assembly- or program-control flag or parameter. The syntax of this command is:

```
RESET  { memory = size | verbose }
```

The first of these two choices allows the user to specify that memory is limited to *size* computer words (the program's default limit being installation-dependant). The second choice (verbose) operates a toggle switch that changes the program's verbosity switch from false (the default value) to true, or vice versa, giving the user some control over how much execution-time information is printed out as the assembly progresses.

12.2.3.3 INCLUDE Command

The INCLUDE command is used to: (i) identify data entities that contribute to the next system matrix (and/or vector) to be assembled, (ii) identify DOF and constraint information for the system matrix and/or vector to be assembled, or (iii) purge any items from the list of previously-included entities. Entities to be included must be stored in datasets on GAL library file(s) in appropriate format(s). ASM currently treats the following types of datasets:

Dataset Type	Description
CSM	Complete summary of the model, a CSM data object
EMT	Element-matrix contributions, an EMT data object
NDT	Nodal DOF and constraint information, an NDT data object
NVT	Right-hand-side system-vector contributions, an NVT data object
SVT	Right-hand-side system vector contributions, an SVT data object

ASM recognizes, but does not treat, the COMPACT-, COMPAXX-, and SKY_MATRIX-type datasets during its information-gathering stage. ASM cannot currently continue with the assembly when these types of datasets are included.

Dataset Type	Description
COMPACT	Assembled-matrix contributions, stored in the upper-triangle-by-columns, COMPACT system-matrix format (see Chapter 15).
COMPAXX	Assembled-matrix contributions, stored in the upper-triangle-by-rows, COMPAXX system-matrix format (see Chapter 15).
SKY_MATRIX	Assembled-matrix contributions, stored in the 'SKY_MATRIX (symmetric) skyline-matrix format (see Chapter 15).

The formal syntax of the INCLUDE command is:

```
INCLUDE    [ /X ]  ldi  { ds_name | seq }
```

```
[ CONTENTS = cntnts ] ++
[ DEFINITION = ldi_def ] ++
[ { EXCEPT | ONLY } = lds { subset | jseq } ] ++
[ ORDER = ldo { ord_set | kseq } ]
```

The command items are summarized in the following table and described in detail following the table.

Item	Item Description
X	Optional qualifier used to rescind one or more previous INCLUDE commands.
ldi	Logical device index for the GAL library containing the entity to be included. If *ds_name* is specified, the entity to be included is stored in dataset *ds_name* (or in the datasets indicated by *ds_name* if *ds_name* contains wildcard characters). If the integer list *seq* is given, the entities to be included are stored in the dataset(s) with sequence number(s) in that list.
CONTENTS ...	Required phrase if the dataset(s) designated in the preceding {*ds_name* | *seq*} phrase are not self-descriptive (do not contain recognized CONTENTS records); this phrase identifies the types of data contained in a non-descriptive datasets and in NVT- and SVT- data objects.
DEFINITION	Optional definition phrase.
EXCEPT	Optional exclusion phrase (not implemented).
ONLY	Optional selection phrase (not implemented).
ORDER	Optional permutation phrase (not implemented).

The optional X qualifier is used with the INCLUDE command to rescind one or more previously-submitted INCLUDE specifications. If the X qualifier is not used, the INCLUDE command tells ASM to examine all of the specifications given on the remainder of the command and to include entities identified there in tables ASM uses to keep track of contributions to and constraints upon the matrix to be assembled. If the X qualifier is used, the INCLUDE command tells ASM to remove all previously-included entities stored in dataset(s) { *ds_name* | *seq* } on library *ldi* from these tables.

The CONTENTS phrase is used to identify the contents of included datasets that are not self-descriptive, and to indicate operations to be performed by ASM with various self-descriptive (data-object) datasets. ASM currently recognizes the following *cntnts* designations.

cntnts Value	Value Description
FORC_NODVEC	The dataset contains RHS contributions, stored as an NVT data object
DIAG_NODVEC	The dataset contains contributions to the system-matrix diagonal, stored as an NVT-object
DISP_NODVEC	The dataset contains prescribed RHS values, stored as an NVT-object

Extensions of ASM are anticipated to process datasets with the following *cntnts* designations:

Future *cntnts* Value	Value Description
FORC_DOFVEC	The dataset contains RHS contributions, stored as an SVT data object
DISP_DOFVEC	The dataset contains prescribed RHS values, stored as an SVT data object

The DEFINITION phrase is required only when the entity being included is a single element-contribution or a set of element-contribution EMT data object(s). The *ldi_def* parameter identifies the GAL library that contains the supplementary element-definition EDT data object(s) for those contributions. The DEFINITION phrase must not be used under any other circumstances.

The EXCEPT phrase has been included in anticipation that ASM will be enhanced to INCLUDE a subset of the contributions stored in the indicated dataset by excluding some of the information therein using the exclusion vector {E} that is stored in dataset subset (or in the dataset(s) identified by sequence number(s) *jseq*) on library *lds*. This capability has not been fully implemented and should not be exercised at this time.

The ONLY phrase has been included in anticipation that ASM will be enhanced to INCLUDE a subset of the contributions stored in the indicated dataset by including only the items designated in the inclusion vector {E} that is stored in dataset subset (or in the dataset(s) identified by sequence number(s) *jseq*) on library lds. This capability has not been fully implemented and should not be exercised at this time.

The ORDER phrase has been included to permit the user to apply a permutation vector {P}, which is stored in dataset *ord_set* (or in the dataset with sequence number *kseq*), on GAL library *ldo*, to modify (within ASM) the information in the dataset(s) to be included via this command. This capability might be used, for example, to introduce an alternate nodal sequencing vector (NOT data object), to re-order the equation system in the DOF-table (NDT data object) being used. This capability has not been fully implemented and should not be exercised at this time. It should be exercised with great caution when it is implemented.

The user must specify a compatible Complete Model Summary Table (CSM data object) via the MODEL command prior to any attempt to INCLUDE anything. A Nodal DOF Table (NDT data object) must also be included in order to assemble anything.

The INCLUDE command (and each of the keywords on the INCLUDE command) may be abbreviated to two characters.

12.2.3.4 OUTPUT Command

The OUTPUT command is used to specify where an assembled system matrix (and/or system vector) is to be saved, or to purge any previously-specified OUTPUT requests from the program's output-request table. The OUTPUT command must be used at least once before the ASSEMBLE command is given and may be used more than once to save the assembled information in more than one output format.

The formal syntax of the OUTPUT command is:

```
OUTPUT [ /X ]   ldi  ds_name   [ FORMAT = format ]
```

The command items are described below.

Item	Item Description
X	Optional qualifier used to rescind one or more previous OUTPUT specifications
ldi	Logical device index for GAL library to receive the assembled matrix or vector
ds_name	Dataset name where the assembled system matrix (or vector) is to be stored (on library *ldi*)
format	The format in which an assembled entity is to be archived: DOFVEC — SVT data object format (for a RHS vector) COMPACT — upper triangle by columns (for a system matrix) COMPAXX — upper triangle by rows (for a system matrix) SKYLINE — SKY_MATRIX format (for a system matrix)

The optional X qualifier is used with the OUTPUT command to rescind previously-specified OUTPUT specifications. If the X qualifier is not used, destination and format specifications are added to a table that ASM interrogates when archiving operations are performed. If the X qualifier is used, the indicated output specifications are removed from that table.

ASM normally receives a variety of contributions and definitions, one of which must be an appropriate Nodal DOF Table (NDT data object). ASM collects the contribution, degree-of-freedom, constraint, and topological data associated with those definitions, analyzes those data, and then assembles the desired system matrix (and/or vector) into a compact, space-efficient, upper-triangle-by-columns format before converting it (when necessary) to the output format requested.

If the user does not include the FORMAT phrase, ASM will save any assembled matrix that it constructs in its default SKY_MATRIX format. Specify FORMAT = DOFVEC for any RHS vector to be assembled.

12.2.3.5 SHOW Command

The optional SHOW command may be used at any time before an assembly to display information about the matrix to be assembled, or to set parameters that control printout of the assembled system matrix (and/or vector). The formal syntax of the SHOW command is:

```
SHOW    { DOF_DATA |
          INCLUDE  |  OUTPUT  |  VECTOR  |
          MATRIX [ DIAGONAL | ++
                   [ COLUMN = ldc { col_set | cseq } ] ++
                   [ ROW = ldr { row_set | rseq } ] ] }
```

The keywords cause actions as noted below.

Keyword	Keyword Description
DOF_DATA	Displays the included DOF Table (NDT data object) when it is accessed (immediately prior to assembling the system matrix and/or vector).
INCLUDE	Displays information about contributions to the matrix to be assembled.
OUTPUT	Displays the library location(s) and output-dataset format(s) specified for the system matrix and/or system vector to be assembled and archived.
VECTOR	Displays the assembled RHS vector (if any).
MATRIX	Displays the diagonal of the assembled matrix, or some or all of the nonzero entries (and their row and column locations) in the entire assembled matrix. The COLUMN and ROW sub-commands are not yet implemented. Since columns and rows of interest cannot be selected, use of the MATRIX option on the SHOW command (without exercising the DIAGONAL option) currently causes the entire matrix to be printed. The number of nonzero entries in anything but the smallest assembled matrix is generally very large, thus SHOW MATRIX generates an enormous amount of printout for most system matrices.

12.2.3.6 ASSEMBLE Command

The ASSEMBLE command causes the assembly of a system matrix and/or a computational RHS vector. Contributions to the matrix (and/or vector), and a suitable Nodal DOF Table (NDT data object), must have been defined (with one or more INCLUDE commands), and the archival-storage destination(s) of the resulting assembled system matrix (and/or vector) must have been defined (with appropriate OUTPUT commands) before the ASSEMBLE command is used.

The formal syntax of the ASSEMBLE command is:

```
ASSEMBLE [ /MATRIX | /VECTOR | /MATRIX,VECTOR | /VECTOR,MATRIX ]
```

The command qualifiers are described below.

Qualifier	Qualifier Description
MATRIX	Optional qualifier specifying that a system matrix is to be assembled (default)
VECTOR	Optional qualifier specifying that a system vector is to be assembled

If neither of these qualifiers is given, the MATRIX option is assumed. If both qualifiers are given, ASM attempts to assemble a system matrix and a system vector.

Typically, the user also enters CLIP *OPEN directives as many times as necessary to open GAL library files containing contribution, constraint, and/or any other kinds of input data required for the assembly work to be done, and as many more times as necessary to open the GAL library file(s) to receive the assembled entities.

Contributions to the system matrix must be specified through at least one use of the INCLUDE command. DOF and constraint information for the system must also be specified through an additional INCLUDE command. When everything that contributes to (and possibly constrains) the system matrix has been specified, the ASSEMBLE command is invoked. With the ASSEMBLE command, ASM first performs a topological analysis by looking at nodal connectivity for contributors to the matrix with that kind of information. ASM uses the results of that analysis to partition its internal workspace and make decisions about the assembly operations. ASM then assembles the system matrix into a space-efficient, compact data structure used for its internal operations.

If the COMPACT format option was selected on an OUTPUT command, ASM saves the assembled matrix in the upper-triangle-by-columns format. If the COMPAXX format option was selected on an OUTPUT command, ASM saves the assembled matrix in the upper-triangle-by-rows format. If the SKYLINE option was selected, ASM transforms the compact-formatted matrix into the default SKY_MATRIX data format and saves it in that form. This data format is also used by default when an option format is not specified. ASM saves the assembled computational RHS vector as an SVT data object if the FORMAT=DOFVEC clause was included on an OUTPUT command.

When the assembly and archiving operations are completed, ASM returns to the user-input command post with all of the currently active INCLUDE, OUTPUT, and SHOW specifications still in place. At this point, the user can enter additional INCLUDE, OUTPUT, and/or SHOW commands to construct a modified version of the system matrix and/or vector that has already been assembled. The user can then issue the ASSEMBLE command again to assemble (from scratch) the modified system matrix (and/or vector), issue the RESTART command to clear the boards before treating a totally new, independent assembly problem, or issue the STOP command (or a RUN command) to terminate the ASM processor. ASM operations are normally terminated with the STOP command.

12.2.3.7 RESTART Command

The RESTART command is used to re-initialize ASM, to start a completely new assembly problem from scratch. The formal syntax of this command is:

```
RESTART
```

ASM responds to the RESTART command by clearing all of its INCLUDE-, OUTPUT-, and SHOW-specification tables by erasing (but not releasing) its local memory and returning to the user-interface command post to await instructions for the next system matrix to be assembled. The RESTART command is usually issued after the successful completion of an assembly operation, but it may be used at any time the user wants to start over.

12.2.3.8 STOP Command

The STOP command is used to terminate ASM operations normally. The formal syntax of this command is:

```
STOP
```

ASM recognizes END and EXIT as synonyms for STOP, and terminates normally if either of these alternates is submitted. ASM also recognizes the RUN command and terminates normally when a command of that form is submitted.

In any event, when ASM terminates normally, active libraries are flushed and closed and information for the next CSM processor (if any) is passed on for further use down the line.

12.2.4 Database Input/Output Summary

12.2.4.1 Input Datasets

A summary of input datasets for processor ASM is given below in Table 12.2-2.

Table 12.2-2 Processor ASM Input Datasets

Dataset	Type	Description
CSM.SUMMARY...*mesh*	CSM	Complete Model Summary dataset for mesh=*mesh*
NODAL.DOF..*cons.mesh*	NDT	Nodal DOF Table (constraint case=*cons*, mesh=*mesh*)
Eltname.DEFINITION...*mesh*	EDT	Element-definition data (optional)
Eltname.Matname...mesh	EMT	Element-contribution data (optional)
NODAL.*Vector.step..mesh*	NVT	RHS-contribution data (optional)

The CSM and NDT datasets are required for any assembly operations. One or more EMT (element-contribution) datasets (with their associated EDT-formatted, element-definition datasets) will be required for assembly of a system matrix and may be used for assembly of a system RHS vector. The names of these datasets are generally determined by element processors (used prior to the assembly process) and are not hard-wired into the ASM processor. One or more NVT-formatted datasets may be required if the user includes right-hand-side contributions and/or specified-value information. The names of these datasets are not hard-wired into ASM either. Specification of how these data are to be interpreted is accomplished via the CONTENTS phrase on the INCLUDE command used to bring them into ASM.

12.2.4.2 Output Datasets

A summary of output datasets that may be produced by processor ASM (depending on the OUTPUT option(s) specified by the user) is given in Table 12.2-3.

Table 12.2-3 Processor ASM Output Datasets

Dataset	Dataset Type	Description
Matrix..step.cons.mesh	COMPACT	Assembled system matrix
Matrix..step.cons.mesh	COMPAXX	Assembled system matrix
Matrix..step.cons.mesh	SKY_MATRIX	Assembled system matrix
SYSTEM.*Vector...mesh*	SVT	Assembled RHS vector

Given compatible input, Processor ASM produces a COMPACT-formatted system-matrix dataset if the OUTPUT command is used with the FORMAT=COMPACT clause (assuming that the ASSEMBLE command is then used with the MATRIX qualifier); and/or a COMPAXX-formatted system matrix if the OUTPUT command is used with the FORMAT=COMPAXX clause; and/or a SKY_MATRIX-formatted system matrix if the OUTPUT command is used with the FORMAT=SKYLINE clause. The name(s) of these datasets are not hard-wired into ASM and can be set by the user.

Given compatible input, Processor ASM produces an SVT data object containing an assembled computational RHS system vector if the OUTPUT command is used with the FORMAT=DOFVEC clause prior to invocation of the ASSEMBLE command with the VECTOR qualifier. The name of this dataset is not hard-wired into ASM and can be anything the user chooses.

12.2.5 Limitations

ASM is currently implemented as a main-memory (in-core) processor. Sufficient main memory must be available to store the following information:

- The entire CSM data object, which contains important problem-size and other information for the system to be assembled;
- A portion of the Nodal DOF Table (NDT data object), which contains DOF-type, constraint-status, and constraint-reference information for the problem at hand;
- A portion of the Element Matrix Table for any EMT data object that contributes to the system to be assembled;
- A portion of the Element Definition Table (EDT data object) for any EMT data object that contributes to the system to be assembled; the EDT data object contains nodal-connectivity

data for those contributions;

- All vectors specified through EXCEPT-, ONLY-, and/or ORDER-clauses on any INCLUDE commands;
- Two integer-type and two floating-point-type vectors that are needed for the COMPACT representation of the assembled system matrix;
- A single N_{eq}-entry (floating point) computational vector, if a vector is to be assembled; plus any (NDOF x NNODES)-entry NODVEC-formatted vectors that contribute to or contain specified values for the RHS vector to be assembled.

Following a successful assembly, ASM frees the memory required for nodal connectivity and other contribution information. If output is requested in the SKY_MATRIX format, this memory (and whatever else is required) will be used to store the skyline representation of the system matrix (which must exist in main memory simultaneously with the compact representation).

For conversion of the upper-triangle-by-columns version of an assembled system matrix to the transposed (upper-triangle-by-rows) format, an additional N_{eq}-entry integer vector is required for storage of pointer information. Integer and floating-point vectors are also required for the nonzero values and their locations in the system matrix. If available memory is too small to contain both versions of the assembled matrix simultaneously, ASM allocates integer- and floating-point workspaces that are as large as possible and forms the desired location and value records block-by-block as required.

An additional N_{eq}-entry integer-type vector is required for conversion of the upper-triangle-by-columns version of an assembled system matrix to the SKY_MATRIX format. A floating-point vector that is large enough to contain (as much as possible of) the active columns of the SKY_MATRIX-formatted matrix is also required: this vector is generally much larger than its compact-format counterparts, since it must accommodate all entries within the profile of the skyline, including zeros. ASM forms the values record in one pass if it can be fully accommodated within the available memory or on a block-by-block basis if it cannot.

Other less serious limitations have been noted throughout the preceding text. These are primarily related to program features that have been projected but not yet fully implemented.

12.2.6 Error Messages

Processor ASM generates about 150 internally-constructed error messages, all of which are as self-explanatory as possible. Care has been taken to detect and explain user errors as early as possible in the data-definition process or, failing that, as soon as possible during the assembly process. Additionally, ASM passes on to the user a number of error messages that are returned to ASM by the CSM*, EDT*, EMT*, NDT*, NOT*, NVT*, and SVT* utilities that ASM uses to access model-summery, element-definition, element-matrix, DOF-Table, nodal-ordering, nodal-vector, and computational-vector information, when problems are encountered there.

The following three messages are typical of those that are produced by the part of ASM that reads and checks information on an INCLUDE command.

```
Sub-Command Keyword Expected...
          Valid Sub-Commands... EXCEPT  ONLY  ORDER
Dataset QWERTY.ZAP Is Not Self-Descriptive (It Has No CONTENTS Record)
ASM Does Not Understand And Cannot Process Type ATAHUALPA Datasets
```

The messages continue to be friendly after the user-input portion of the assembly operation is finished and ASM starts checking the included entities for compatibility and completeness. The following are four typical messages that might be generated during the pre-assembly, data-consistency-checking phase (after the user issues the ASSEMBLE command).

```
Dataset CAVA.BIEN Contributes FULSYM-Formatted Data
ASM Cannot Process FULSYM-Type Contributions At This Time !!!
!!! DOF_DATA Have Not Been Specified !!!
??? No OUTPUT Specifications ???
```

The more urgently punctuated messages are generally produced when errors that inhibit the assembly operation are encountered. The most serious of these might be followed by the

```
!!! ASM ASSEMBLY OPERATIONS DISCONTINUED !!!
```

message, which ASM tries to avoid issuing but must when it cannot determine what to do.

12.2.7 Examples and Usage Guidelines

Remember ASM's limitations and refrain from using ASM for extremely large problems and from using program features that have not been implemented.

The following generic commands constitute the minimum required to assemble a system matrix.

*OPEN	*ldi file_name*	. Open a GAL file
INCLUDE	= *ldi model_summary_table*	. Model Summary Table
INCLUDE	= *ldi* { *NDT_dataset* | *seq_number* }	. Nodal DOF Table
INCLUDE	= *ldi* { *contribution_dataset* | *seq_number* }	. Element/Vector contributions
OUTPUT	= *ldi output_dataset*	. Output destination
ASSEMBLE		. Go-Do-It command
STOP		. Exit ASM

12.2.7.1 Example 1

The following commands might be used in a more typical situation to assemble a constrained system matrix and an RHS vector, to save the matrix in two output formats, and to save the RHS computational system vector as an SVT data object.

*open	1 HSCT.DBC		Open GAL file
include	1 CSM.SUMMARY		Model Summary table
incl	1 NODAL.DOF..1		Nodal DOF Table (NDT obj)
inc	1 E*.MATL_STIFFNESS	DEFINITION 2	Element (EMT) contributions
In	1 NODAL.EXT_FORC.1	CONTENTS=FORC_S	RHS vector contributions
output	2 CIEL.BLEU.1	FORMAT=SKYLINE	Output matrix destination
outp	2 TOY.OTA.1	FORMAT=COMPACT	Output matrix destination
Out	2 RHS.VEC.1	format = DOFVEC	Output vector destination
assem	/matrix,vector		Assemble matrix & vector
stop			Exit ASM

12.2.8 References

[1] Stanley, G. M., Loden, W., Regelbrugge, M., Stehlin, B., and Wright, M., *The Computational Structural Mechanics (CSM) Testbed User's Manual: New Lockheed Processors*, Lockheed Contract Report, May 1989.

[2] Felippa, C. A., *The Computational Structural Mechanics Testbed Architecture: Volume II: Directives*, NASA CR–178385, February 1989.

[3] Stanley, G. M. and Swenson, L., *HDB: Object-Oriented Database Utilities for COMET-AR*, NASA Computational Structural Mechanics (CSM) Contract Report, August 1992

12.3 Processor ASMs (Matrix Assembler)

12.3.1 General Description

Processor ASMs is a general purpose, out-of-core, linear equation assembly processor using the skyline storage format (SMT data object class).

ASMs includes special provisions for the particular needs of the AR environment, notably:

- ***Partial assembly.*** Assemble only new columns which are appended to a previously assembled/factored matrix (required by the h_s-refinement technique).
- ***Preconditioning matrix assembly.*** Assemble only the block diagonal terms for each mesh to be used in a Preconditioned Conjugate Gradient (PCG)[1] iteration solver (only when h_s-refinement technique is being used).
- ***Compute coupling stiffness terms for h_s-refinement.*** ASMs computes the coupling stiffness terms between the superposed mesh and the underlying elements using the S-Interpolation Property (SIP)[2] and can also update block diagonal stiffness of an underlying mesh to account for changes in geometry presentation between superposed mesh (fine) and an underlying mesh (coarse).

ASMs employs an out-of-core block assembly algorithm whose memory buffer is dynamically allocated at run time with a size controlled by the user.

Processor ASMs is typically invoked by a high-level AR control procedure, such as AR_CONTROL (via procedure L_STATIC_1), in an adaptive refinement iterations loop.

12.3.2 Command Summary

Processor ASMs follows standard COMET-AR command interface protocol. A summary of ASMs commands is given in Table 12.3-1.

Table 12.3-1 Processor ASMs Command Summary

Command Name	Function	Default Value
SET BUFFER_SIZE	Specifies size in float words of the memory buffer used by ASMs to store each of the system matrix blocks.	524288 (2MB single, 4MB double)
SET CONSTRAINT_SET	Specifies constraint-set number.	1

1. See Section 12.4, *Iterative Linear Equation Solver* for details about the PCG algorithm.
2. See "Superposition-based (h_s) Adaptive Refinement of Shell Structures," Lockheed Contract Report for NASA CSM Task 15, November 1991.

Table 12.3-1 Processor ASMs Command Summary (Continued)

Command Name	Function	Default Value
SET FIXED_FRAME	Fixed frame flag for h_s-refinement. When the computation frame is fixed in the model, ASMs can compute the stiffness coupling terms, using the SIP method, without applying any frame transformation to the stiffness arrays. This will greatly increase the performance of ASMs.	NO
SET LDIC	Specifies logical device index of computational database.	1
SET LDIE	Specifies logical device index of element matrices database (required only for iterative solutions).	2
SET LDIS	Specifies logical device index of system database.	3
SET LOAD_SET	Specifies load-set number.	1
SET MESH	Specifies mesh number.	0
SET STEP	Specifies load step number.	0
ASSEMBLE/*system_entity*	Assembled system matrix or vector.	

12.3.3 Command Definitions

12.3.3.1 ASSEMBLE Command

This is the "go" command for processor ASMs. It causes ASMs to assemble the proper system entity specified by the *system_entity* qualifier.

Command syntax:

ASSEMBLE/*system_entity*

where

<table>
<tr><th>Parameter</th><th colspan="2">Description</th></tr>
<tr><td>system_entity</td><td colspan="2">Type of system entity to assemble (default value MATRIX):
VECTOR—assemble the right hand side load vector
MATRIX/option/update—assemble the system matrix. The qualifiers option and update are relevant only in the context of h_s-refinement and are described as follows:</td></tr>
<tr><td></td><td>Parameter</td><td>Description</td></tr>
<tr><td></td><td>option</td><td>DIRECT—assemble the full matrix including coupling terms between superposed and underlying meshes; to be factored by a direct equation solver.
PRECONDITIONER—assemble only the block diagonal stiffness, for each mesh, to create a preconditioning matrix for the PCG[a] iterative solver.</td></tr>
<tr><td></td><td>update</td><td>UPDATE_STIFFNESS—update the underlying mesh stiffness matrix (will need full factorization)
NO_UPDATE—do not update the underlying mesh stiffness matrix (will require only PARTIAL factorization)</td></tr>
</table>

a. See Section 12.7, *Direct Linear Equation Solver*, for details

12.3.3.2 SET BUFFER_SIZE Command

This command defines the size of memory buffer used by ASMs to hold each assembled system matrix columns block.

Command syntax:

SET BUFFER_SIZE = *size*

where

Parameter	Description
size	Matrix block size (in floating precision words) (default value: 524288)

12.3.3.3 SET CONSTRAINT_SET Command

This command defines the constraint set number associated with the element, nodal, and system data. This number should appear as the second cycle number in names of all datasets.

Command syntax:

SET CONSTRAINT_SET = *conset*

where

Parameter	Description
conset	Constraint set number (default value: 1)

12.3.3.4 SET FIXED_FRAME Command

Sets the fixed-frame flag to the appropriate state for h_s-refinement solution. ASMs computes the coupling stiffness by using the "S-Interpolation Property" (SIP) to account for the coupling terms between the superposed mesh and the underlying mesh. This process involves interpolation of stiffness matrices which need to be transformed into a global/fixed coordinate system before and after the interpolation, unless a fixed computational frame is used throughout the mesh.

Command syntax:

SET FIXED_FRAME = *flag*

where:

Parameter	Description
flag	Fixed computational frame flag (default value: NO)

12.3.3.5 SET LDIC Command

This command defines the logical device index for the central database.

Command syntax:

SET LDIC = *ldic*

where

Parameter	Description
ldic	Logical device index (default value: 1)

12.3.3.6 SET LDIE Command

This command defines the logical device index for the element matrices database.

Command syntax:

SET LDIE = *ldie*

where

Parameter	Description
ldie	Logical device index (default value: 2)

12.3.3.7 SET LDIS Command

This command defines the logical device index for the system matrices database.

Command syntax:

SET LDIS = *ldis*

where

Parameter	Description
ldis	Logical device index (default value: 3)

12.3.3.8 SET LOAD_SET Command

This command defines the load set number associated with the element, nodal, and system data. This number should appear as the first cycle number in names of all datasets.

Command syntax:

SET LOAD_SET = *ldset*

where

Parameter	Description
ldset	Load set number (default value: 1)

12.3.3.9 SET MESH Command

This command defines the mesh number for the mesh to be processed. This number should appear as the third cycle number in names of all datasets.

Command syntax:

SET MESH = *mesh*

where

Parameter	Description
mesh	Mesh to be assembled (default value: 0)

12.3.3.10 SET STEP Command

This command defines the solution step number (for nonlinear analyses only) associated with the element, nodal, and system data. This number, if defined, should appear as the first cycle number in names of all datasets.

Command syntax:

SET STEP = *step*

where

Parameter	Description
step	Solution step number. (default value: 0)

12.3.4 Database Input/Output

12.3.4.1 Input Datasets

A summary of input datasets required by Processor ASMs is given in Table 12.3-2.

Table 12.3-2 Processor ASMs Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset
NODAL.DOF..*conset.mesh*	NDT	Nodal DOF dataset

Table 12.3-2 Processor ASMs Input Datasets (Continued)

Dataset	Class	Contents
†NODAL.COORDINATE...*mesh*	NCT	Nodal coordinate dataset
†NODAL.TRANSFORMATION...*mesh*	NTT	Nodal transformation dataset
NODAL.EXT_FORCE.*ldset..mesh*	NVT	Nodal external load dataset
NODAL.SPEC_DISP.*ldset..mesh*	NVT	Nodal specified displacement dataset
EltNam.DEFINITION...*mesh*	EDT	Element definition dataset
†*EltNam*.REFINEMENT...*mesh*	ERT	Element refinement dataset
†*EltNam*.INTERPOLATION...*mesh*	EIT	Element interpolation dataset
EltNam.MATL_STIFFNESS...*mesh*	EMT	Element stiffness matrix dataset (in file LDIE)
EltNam.LOAD...*mesh*	ELT	Element loads datasets

†h_s-refinement only

12.3.4.2 Output Datasets

A summary of output datasets created by Processor ASMs is given in Table 12.3-3.

Table 12.3-3 Processor ASMs Output Datasets

Dataset	Class	Contents
†*EltNam*.COUP_STIFF...*mesh*	EAT	Element coupling matrices dataset (in file LDIE)
†*EltNam*.TCGHAT...*mesh*	EAT	Element updated rotations transformation (accounts for geometry updates in superposed mesh, in file LDIC)
SYSTEM.VECTOR.*ldset..mesh*	SVT	System load vector
COLUMN.HEIGHT...*mesh*	SMT	Columns heights (in file LDIS)
DIAGONAL.ADRESSES...*mesh*	SMT	Diagonal elements pointers (in file LDIS)
STRUCTURE.MATL_STIFFNESS...*mesh*	SMT	The assembled system matrix (in file LDIS)

†—h_s-refinement only

12.3.5 Limitations

12.3.5.1 SMT Data Structure

ASMs currently outputs the assembled matrix as an SMT type data structure recognized by the SKYs solver. Other solvers within COMET-AR require different types of input data structures (such as COMPAXX format) which cannot be produced by ASMs.

12.3.5.2 Multi-Point Constraints

ASMs currently does not include any capabilities for applying MPCs to the assembled matrix, and thus cannot be used as an assembly processor within constraint-based (h_c) adaptive refinement.

12.3.6 Error Messages

ASMs contains extensive error checking. Most of the error messages printed by ASMs are self-explanatory messages and aim to help the user correct mistakes. Some of the errors may occur at code levels below ASMs (e.g., HDB, DB, GAL, etc.) and ASMs describes those errors to the best of its ability.

The following summarizes error messages related to user interface problems produced by ASMs.

Index	Error Message	Cause	Recommended User Action
1	Unknown SET *variable name* encountered in ASMs.	ASMs user interface cover encountered an unrecognized SET variable name.	Check spelling of *variable name* in CLIP procedure.
2	Unknown *command* encountered in ASMs.	ASMs user interface cover encountered an unrecognized command.	Check spelling of *command* in CLIP procedure.
3	Old/new *dataset name* could not be opened in *routine name* —or— problem in *routine name*: missing *dataset name* for the following *element name*.	ASMs could not open a certain dataset.	1. Check execution log file; look for error produced by processors prior to ASMs execution. 2. Verify *dataset name* using HDBprt processor. 3. Make sure that all required input datasets are present in the database file.
4	*Dataset name* could not be closed in *routine name*.	ASMs could not close a certain dataset.	1. Check execution log file; look for errors previously produced by processor ASMs. 2. Verify that ASMs is the only processor accessing the database file. (Is ARGx being used in the same directory?)
5	*Dataset name* access problem encountered in *routine name* —or— could not get/put/add/update/*attribute name* to *dataset name* in *routine name*.	ASMs could not get/put an attribute from/to the dataset name table.	Verify that the particular dataset contains attributes required by ASMs.
6	Could not allocate memory for array name in *routine name*.	Malloc memory allocation problem encountered in DBmem level.	Check with your system manager. You may need permission to access more memory.

In addition to the above generic messages, ASMs will print any relevant information regarding the problem to assist the user in correcting the error. A full trace-back printout of error messages will follow the first message, and ASMs will attempt to terminate its execution as cleanly as possible (closing opened datasets, releasing memory allocations, etc.).

12.3.7 Examples and Usage Guidelines

12.3.7.1 Example 1: Assembling the Load Vector

```
RUN ASMs
              SET MESH              = 1
              ASSEMBLE/VECTOR
STOP
```

In this example, ASMs will assemble the load vector for mesh 1, load set 1, and constraint set 1 assuming standard naming conventions for the input files. Nodal external loads will be transferred into a system vector format and element-prescribed displacement contributions will be computed and assembled.

12.3.7.2 Example 2: Assembling the Stiffness Matrix

```
RUN ASMs
              SET MESH              = 1
              SET BUFFER_SIZE       = 102400
              ASSEMBLE/MATRIX
STOP
```

In this example, ASMs will assemble the stiffness matrix of mesh 1, using 102400 words of physical memory for the assembly buffer. The assembled matrix, in SMT format, will be stored in the standard system database (logical device index 3).

12.3.7.3 Example 3: Assembling a Preconditioner Matrix for h_s-refinement

```
RUN ASMs
              SET MESH                      = 1
              SET FIXED_FRAME               = ON
```

```
ASSEMBLE/MATRIX/PRECONDITIONER/NO_UPDATE

STOP
```

In this example, ASMs will assemble the block diagonal stiffness matrix of mesh 1 and append this block to an existing system matrix (currently containing the mesh 0 factored matrix). The assembled matrix, in SMT format, will be stored in the standard system database (logical device index 3).

In addition, ASMs will compute the mesh 1 element coupling stiffness, assuming that the computational frame is fixed throughout the mesh, which will be stored as an EAT dataset in the standard element database (logical device index 2).

Finally, since the NO_UPDATE qualifier is set, ASMs will not update any stiffness term associated with mesh 0, ignoring the effects of the changes in geometry modeling between mesh 0 and mesh 1.

12.3.8 References

None.

12.4 Processor ITER (Iterative Linear Equation Solver)

12.4.1 General Description

Processor ITER is a general purpose, iterative, linear equation solver employing the preconditioned conjugate gradients (PCG) scheme with a partial Crout (LDU) factorization of the stiffness matrix as a preconditioner.

The PCG technique enforces the K-orthogonality condition (orthogonality with respect to the stiffness matrix) on successive solution increments and employs a simple Line Search (LS) technique to minimize the solution errors (in the energy norm) during each iteration cycle.

PCG techniques are guaranteed to converge within N_{eq} iterations when applied to a symmetric positive definite quadratic form (such as the strain energy function) provided that a symmetric positive definite preconditioner is employed.

ITER uses the partially factored stiffness matrix (in COMPAXX storage format, see Chapter 15, *Database Summary*) as a preconditioning matrix. The preconditioner matrix is obtained by applying Crout factorization to that form of the assembled matrix without adding the additional below-the-profile terms as traditionally done by conventional solvers. The partially factored matrix occupies the identical storage space as the unfactored matrix, but is only an approximation for the actual factored matrix.

A partially factored matrix may lose its positive-definiteness, easily detected during the factorization step by monitoring negative diagonal terms in the factored matrix. To ensure that the preconditioner is indeed positive definite matrix, ITER employs a technique called diagonal scaling, where diagonal terms of the stiffness matrix **K** are scaled by a small factor as shown in the following equation:

$$\overline{\boldsymbol{K}} \leftarrow \boldsymbol{K} + \alpha(diag(\boldsymbol{K}))$$

The scaling factor α is sufficient to ensure that the partial factorization of $\overline{K}$ is indeed positive definite. ITER may require several factorization attempts to fine-tune the value of α.

Processor ITER is typically invoked by a high-level AR control procedure, such as AR_CONTROL_1 (via procedure L_STATIC_1), in an adaptive refinement iteration loop

12.4.2 Command Summary

Processor ITER follows standard COMET-AR command interface protocol. A summary of ITER commands is given below in Table 12.4-1.

Table 12.3-1 Processor ITER Command Summary

Command Name	Function	Default Value
SET CONSTRAINT_SET	Specifies constraint-set number	1
SET CONV_TOL	Convergence tolerance	10^{-6}
SET LDIC	Specifies logical device index of computational database	1
SET LDIS	Specifies logical device index of system database	3
SET LOAD_SET	Specifies load-set number	1
SET MAX_ITER	Specifies maximum allowable number of iterations	$2\ N_{eq}$
SET MESH	Specifies mesh number	0
SET STEP	Specifies load step number	0
SET SCALE_FACTOR	Initial diagonal terms scaling factor	0.005
FACTOR	Partially factor the mesh to generate the preconditioner matrix	
SOLVE	Obtain a solution using PCG iterations	

12.4.3 Command Definitions

12.4.3.1 FACTOR Command

This is the "go" command for processor ITER's factorization stage. It causes ITER to generate the preconditioner matrix for the specified mesh by using diagonal scaling and partial Crout factorization of the system matrix (stored in the compact-transpose COMPAXX format).

Command syntax:

FACTOR

12.4.3.2 SOLVE Command

This is the "go" command for processor ITER's iterative solution stage. It causes ITER to compute the solution vector using the PCG iteration technique.

Command syntax:

SOLVE

12.4.3.3 SET CONSTRAINT_SET Command

This command defines the constraint set number associated with the element, nodal, and system data. This number should appear as the second cycle number in names of all datasets.

Command syntax:

SET CONSTRAINT_SET = *conset*

where

Parameter	Description
conset	Constraint set number. (default value: 1)

12.4.3.4 SET CONV_TOL Command

This command defines the convergence tolerance for the PCG iterations with respect to energy error norm.

Command syntax:

SET CONV_TOL = *contol*

where

Parameter	Description
contol	Convergence tolerance. (default value: 10^{-6})

12.4.3.5 SET LDIC Command

This command defines the logical device index for the computational database.

Command syntax:

SET LDIC = *ldic*

where

Parameter	Description
ldic	Logical device index. (default value: 1)

12.4.3.6 SET LDIS Command

This command defines the logical device index for the system database.

Command syntax:

SET LDIS = *ldis*

where

Parameter	Description
ldis	Logical device index. (default value: 3)

12.4.3.7 SET LOAD_SET Command

This command defines the constraint set number associated with the element, nodal, and system data. This number should appear as the first cycle number in names of all datasets.

Command syntax:

SET LOAD_SET = *ldset*

where

Parameter	Description
ldset	Load set number (default value: 1)

12.4.3.8 SET MAX_ITER Command

This command defines the maximum allowable number of PCG iterations.

Command syntax:

SET MAX_ITER = *maxiter*

where

Parameter	Description
maxiter	Maximum number of iterations (default value: 2 N_{eq})

12.4.3.9 SET MESH Command

This command defines the mesh number for the system equations to be processed. This number should appear as the third cycle number in names of all datasets.

Command syntax:

SET MESH = *mesh*

where

Parameter	Description
mesh	Mesh to be solved (default value: 0)

12.4.3.10 SET STEP Command

This command defines the solution step number (for nonlinear analyses only) associated with the element, nodal, and system data. This number, if defined, should appear as the first cycle number in names of all datasets.

Command syntax:

SET STEP = *step*

where

Parameter	Description
step	Solution step number (default value: 0)

12.4.3.11 SET SCALE_FACTOR Command

This command defines the initial diagonal scaling factor for the partial factorization stage.

Command syntax:

SET SCALE_FACTOR = *factor*

where

Parameter	Description
factor	Initial diagonal scaling factor (default value: 0.005)

12.4.4 Database Input/Output

12.4.4.1 Input Datasets

A summary of input datasets required by Processor ITER is given in Table 12.4-2.

Table 12.4-2 Processor ITER Input Datasets

Dataset	Class	Contents	
CSM.SUMMARY...*mesh*	CSM	Model summary dataset	
SYSTEM.VECTOR.*ldset..mesh*	SVT	System load vector	
STRUCTURE.MATL_STIFFNESS...*mesh*	System Matrix	Assembled system matrix (in file LDIS). Records description:	
		Record Name	**Description**
		COLLTH	Column (row) heights
		COLPTR	Diagonal elements pointers
		ROWS	Identity of non-zero elements in each row
		DIAG	Diagonal terms
		COEFS	Off-diagonal non-zero terms

12.4.4.2 Output Datasets

A summary of output datasets created by Processor ITER is given in Table 12.4-3.

Table 12.4-3 Processor ITER Output Datasets

Dataset	Class	Contents	
CSM.SUMMARY...*mesh*	CSM	Model summary table	
SYSTEM.VECTOR.*ldset..mesh**	SVT	System solution vector	
STRUCTURE.MATL_STIFFNESS...*mesh**	System Matrix	Assembled system matrix (in file LDIS). Records description:	
		Record Name	**Description**
		D	D of Crout LDL^T decomposition
		L	L of Crout LDL^T decomposition
		ROWS	Identity of non-zero elements in each row
		DIAG	Diagonal terms
		COEFS	Off-diagonal non-zero terms

*—created record

12.4.5 Limitations

12.4.5.1 Memory Limitation

ITER may require a large number of iterations to converge. Typically this number is in the range [1%-10%] of the total number of equations in the system, depending on the condition number of the system matrix. As a result of the large number of computations, it is important to keep all data required by ITER in the physical memory of the computer so that no I/O will be performed during the solution phase.

This will require two system matrices, the preconditioner and the stiffness matrix, to simultaneously reside in memory. Because a COMPAXX storage format is used (which typically requires an order of magnitude less storage than the skyline (SKY-MATRIX) format) relatively large problems can be solved (e.g., over 150,000 DOFs problems were solved successfully on a CONVEX computer).

12.4.6 Error Messages

ITER contains extensive error checking. Most of the error messages printed by ITER are self-explanatory and aim to help the user correct mistakes. Some of the errors may occur at code levels below ITER (e.g., HDB, DB, GAL, etc.), and ITER describes those errors to the best of its ability.

The following summarizes error messages related to user interface problems produced by ITER.

Index	Error Message	Cause	Recommended User Action
1	Unknown SET *variable name* encountered in ITER.	ITER user interface cover encountered an unrecognized SET variable name.	Check spelling of *variable name* in CLIP procedure.
2	Unknown *command* encountered in ITER.	ITER user interface cover encountered an unrecognized command.	Check spelling of *command* in CLIP procedure.
3	*Old/new dataset name* could not be opened in *routine name*.	ITER could not open a certain dataset.	1. Check the execution log file; look for error produced by processors prior to ITER execution. 2. Verify dataset name using the HDBprt processor. 3. Make sure that all required input datasets are present in the database file.
4	*Dataset name* could not be closed in *routine name*.	ITER could not close a certain dataset.	1. Check the execution log file; look for errors previously produced by processor ITER. 2. Verify ITER is the only processor accessing the database file. (Is ARGx being used in the same directory?)
5	*Dataset name* access problem encountered in *routine name* —or— could not get/put/add/update/*attribute name* to *dataset name* in *routine name*.	ITER could not get/put an attribute from/to the dataset name table.	Verify that the particular dataset contain attributes required by ITER.
6	Could not allocate memory for *array name* in *routine name*	Malloc memory allocation problem encountered in DBmem level	Check with your system manager; you may need permission to access more memory.
7	Zero or negative diagonal term encountered in FACTOR, Equation number = *eqn.*	The assembled matrix is a non-positive-definite matrix.	Check your model's element connectivity and boundary conditions. ARGx may be very useful in this regard.

In addition to the above generic messages, ITER will print any relevant information regarding the problem to assist the user in correcting the error. A full trace-back printout of error messages will follow the first message, and ITER will attempt to terminate its execution as cleanly as possible (closing opened datasets, releasing memory allocations, etc.).

12.4.7 Examples and Usage Guidelines

12.4.7.1 Example 1: Basic Operation

```
RUN ITER
                SET MESH          = 1
                SET CONV_TOL      = 10-7
                SET MAX_ITER      = 1000
                FACTOR
                SOLVE
STOP
```

In this example, the assembled matrix of mesh 1 is factored followed by an iterative solution for mesh 1 displacement field using 10^{-7} as the solution energy error norm tolerance and allowing up to 1000 iterations.

12.4.8 References

None.

12.5 Processor PVSOLV (Direct Linear Equation Solver)

Documentation on this NASA-developed processor will appear in a future release of the User's Manual.

12.6 Processor SKY (Direct Linear Equation Solver)

12.6.1 General Description

Processor SKY is designed to complement the processors ASM and COP, and performs three major functions: (i) factorization of a symmetric system matrix; (ii) solution of a linear system of equations (given a factored system matrix and a computational system vector); and (iii) matrix/vector multiplication (given a system matrix and a computational system vector). In all cases the system matrix is stored in the SKY_MATRIX skyline format and the computational system vector is stored as an SVT data object (Reference [1]).

12.6.2 Processor Command Summary

Like all other COMET-AR processors, the SKY processor can interpret two types of commands: (i) general CLIP directives and (ii) processor specific commands. Directives are described in Reference [2] and are used in the usual manner to open, close, and interrogate a GAL database library. The commands unique to the SKY processor are summarized in Table 12.6-1.

Table 12.6-1 SKY Command Summary

Command Name	Function
FACTOR	Factor a SKY_MATRIX formatted matrix
SOLVE	Solve equations with a factored SKY_MATRIX matrix
MULTIPLY	Multiply a SKY_MATRIX matrix by an SVT data object
STOP	Exit the SKY processor

Each of the above SKY commands can be invoked independently provided that all input is of the correct data type. Since the solve operation needs a factored SKY_MATRIX formatted matrix, the usual calling sequence is a FACTOR command followed by a SOLVE command. A status data record is associated with each SKY_MATRIX formatted matrix to indicate whether the matrix has been factored.

12.6.3 Command Glossary

Arguments for each command are composed of GAL dataset identifiers representing vectors and matrices stored as SVT data objects and SKY_MATRIX formats. Each dataset identifier is composed of a logical device index (*ldi*), a dataset name (*dsn*), and an optional data record name (*recn*). If a data record name is omitted, a default record name is used. (Default argument values will be discussed in the individual command descriptions.) Command arguments for matrix and vector dataset identifiers are separated by the symbols '*' (star) and '->' (arrow). The star separates an input matrix from an input vector while the arrow points to the output vector or

matrix of the specified operation. All commands except STOP may be abbreviated to the first three letters of the command word.

12.6.3.1 FACTOR Command

The FACTOR command performs an in-core factorization of a sparse, skyline stored, symmetric matrix. Syntax for the FACTOR command requires naming the input and an output matrices as follows:

FACTOR *k_ldi k_dsn k_rec -> kf_ldi kf_dsn kf_rec*

where

Item	Item Description	Item Default Value
k_ldi	Logical device index of unfactored matrix	—None—
k_dsn	Dataset name of unfactored matrix	—None—
k_recn	Record name of unfactored matrix	MATRIX
kf_ldi	Logical device index of factored matrix	—None—
kf_dsn	Dataset name of factored matrix	—None—
kf_recn	Record name of factored matrix	MATRIX

Required argument items to identify each matrix are *ldi* and *dsn*. The data record name *recn* is optional, and assumes the name MATRIX if omitted. Both the input and output matrices can assume the same name, in which case the output matrix overwrites the input matrix.

Before performing the factor operation, SKY checks if the current status value of the input matrix is UNFACTORED. If the status value is not UNFACTORED, a warning message is printed before the factorization begins.

Associated with each SKY_MATRIX formatted matrix is an integer-valued array, called DIAG_POINTER, indicating where each column of the matrix is located. For more information regarding SKY_MATRIX formatted matrix data structures refer to Chapter 15, *Database Summary*.

12.6.3.2 SOLVE Command

The SOLVE command solves an unconstrained linear system of equations where a full right-hand side (RHS) system vector is prescribed and the input equation matrix has been factored using the FACTOR command provided by SKY. The SOLVE command syntax requires naming the two input and single output (LHS) dataset identifiers as follows.

SOLVE *kf_ldi kf_dsn kf_rec * rhs_ldi rhs_dsn rhs_recn -> lhs_ldi lhs_dsn lhs_recn*

The command arguments are identified below.

Item	Item Description	Item Default Value
kf_ldi	Logical device index of factored matrix	—None—
kf_dsn	Dataset name of factored matrix	—None—
kf_recn	Record name of factored matrix	MATRIX
rhs_ldi	Logical device index of RHS vector	—None—
rhs_dsn	Dataset name of RHS vector	—None—
rhs_recn	Record name of RHS vector	DATA.1
lhs_ldi	Logical device index of LHS vector	—None—
lhs_dsn	Dataset name of LHS vector	—None—
lhs_recn	Record name of LHS vector	DATA.1

If the status value of the input matrix is UNFACTORED, then the SOLVE operation is aborted. The input and the solution vectors can have the same name, in which case the output vector overwrites the input vector. The default record name for the RHS and LHS vectors is DATA.1, while the default name for the factored matrix is MATRIX.

12.6.3.3 MULTIPLY Command

The MULTIPLY command multiplies a SKY_MATRIX formatted matrix by a full system vector (LHS). The MULTIPLY command syntax requires naming the two input and single output (RHS) dataset identifiers as follows.

*MULTIPLY kf_ldi kf_dsn kf_rec * lhs_ldi lhs_dsn lhs_recn -> rhs_ldi rhs_dsn rhs_recn*

The command arguments are identified as follows.

Argument	Argument Description	Argument Default Value
kf_ldi	Logical device index of factored matrix	—None—
kf_dsn	Dataset name of factored matrix	—None—
kf_recn	Record name of factored matrix	MATRIX
lhs_ldi	Logical device index of LHS vector	—None—
lhs_dsn	Dataset name of LHS vector	—None—
lhs_recn	Record name of LHS vector	DATA.1
rhs_ldi	Logical device index of RHS vector	—None—
rhs_dsn	Dataset name of RHS vector	—None—
rhs_recn	Record name of RHS vector	DATA.1

The input and the solution vectors can have the same name, in which case the output vector overwrites the input vector. The default record name for the RHS and LHS vectors is DATA.1, while the default name for the matrix is MATRIX.

12.6.3.4 STOP Command

The STOP command terminates the current execution of the SKY processor. This command requires no arguments.

```
STOP
```

Invoking this command properly closes all GAL libraries. Executing another COMET-AR processor with the RUN *processor_name* command has the same effect as using the STOP command. It is suggested that all processors be terminated with an explicit STOP command for clarity.

12.6.4 Database Input/Output Summary

12.6.4.1 Input Datasets

The SKY processor makes no assumptions regarding dataset names, as long as they follow standard naming conventions (see Table 12.6-2). Matrices are assumed to be stored using the SKY_MATRIX format and system vectors as SVT data objects. For more information regarding SKY_MATRIX and SVT data structures refer to Chapter 15, *Database Summary*.

Table 12.6-2 Processor SKY Input Datasets

Dataset Name	Dataset Type	Description
Matrix.step.cons.mesh	SKY_MATRIX	Assembled SKY_MATRIX formatted matrix
SYSTEM.*Vector...mesh*	SVT	DOF-oriented SVT data object

12.6.4.2 Output Datasets

The output datasets created by SKY follow the same conventions as the input datasets:

Table 12.6-3 Processor SKY Output Datasets

Dataset Name	Dataset Type	Description
Matrix.step.cons.mesh	SKY_MATRIX	Assembled SKY_MATRIX formatted matrix
SYSTEM.*Vector...mesh*	SVT	DOF-oriented SVT data object

A factored matrix created using the FACTOR command also produces data records summarizing the factorization. The data records listed in Table 12.6-4 are created following the FACTOR command.

Table 12.6-4 Processor SKY Output Records

Data Record Name	Record Type	Description
COEF_DET.0	Float	Coefficient of the determinant
EXP10_DET.0	Int	Exponent of determinate to the base 2
NEG_ROOTS.0	Int	Number of negative matrix diagonal elements
SIGN_DET.0	Int	Sign of the determinant

In addition to these output records, SKY also creates the macrosymbols listed in Table 12.6-5.

Table 12.6-5 Processor SKY Macrosymbols

Macrosymbol Name	Type	Description
coef_det	Float	Coefficient of the determinant
exp10_det	Int	Exponent of determinant to the base 2
num_neg	Int	Number of negative matrix diagonal elements
sign_det	Int	Sign of the determinant

12.6.5 Limitations

12.6.5.1 Main Memory Limitations

Each of the SKY commands requires that the upper triangular portion of the SKY_MATRIX formatted matrix reside in main memory. This requirement is SKY's most important limitation. Commands SOLVE and MULTIPLY also require memory space to store SVT data objects (in addition to a small work space). The total memory space required by each command is summarized below.

Operation	Memory Space Requirement
FACTOR	*Neq**(4**iprc* + 1) + *iprc***nmat* + 1
SOLVE	*Neq**(3**iprc* + 1) + *iprc***nmat* + 1
MULTIPLY	*Neq**(4**iprc* + 1) + *iprc***nmat* + 1

where *Neq* is the number of equations, *iprc* is a precision flag (with a value of 1 or 2), and *nmat* is the number of entries stored in the matrix skyline.

12.6.5.2 Allowed Data Structures

SKY assumes that all vectors are stored as SVT data objects. Although SKY can read Nodal Vector Table (NVT) data objects, it presently outputs all vector results as SVT data objects. It is recommended that the capabilities of COP and ASM be used to contract an NVT data object to an SVT data object, and to expand an SVT data object to an NVT data object. Failure to do this might result in conflicts between SKY and other COMET-AR processors.

12.6.5.3 Case Sensitivity

SKY specific commands and dataset names are case sensitive. All input is expected in uppercase.

12.6.6 Error Messages

The most commonly occurring error messages printed by SKY are presented in Table 12.6-6. Each message has an associated probable cause and a recommended action.

Table 12.6-6 Processor SKY Error Messages

Index	Error Message	Probable Cause	Recommended Action
1	Invalid command option	The SKY processor does not recognize the command.	Check spelling of input command and case.
2	Command syntax error	The correct SKY command has been specified, however the separator symbol (star or arrow) has been improperly placed.	Use correct input format.
3	Dataset does not exist	Unable to find one of the specified datasets in the database.	Check spelling and status of the logical device index.
4	Invalid data type	There is a data type conflict between specified datasets.	Check type of all data records specified.
5	Dataset length mismatch	Input matrix and output matrix have different lengths.	Check if length of the matrices are different; also inspect the load vector.
6	Invalid vector length	Input and output vectors have different lengths.	Make sure that input and output vectors are consistent.
7	Blank common too small	Not enough memory.	Increase the size of blank common.
8	Core too small	Not enough memory.	Increase the size of blank common.
9	Matrix is unfactored	Attempt to use an unfactored matrix with the SOLVE operation.	Factor or use different input matrix.

12.6.7 Example and Usage Guidelines

The following is a typical command script for the SKY processor.

```
RUN SKY

   FACTOR 1 SKY_INPUT -> 2 SKY_OUTPUT

   SOLVE  2 SKY_OUPUT  * 1 RHS_VECTOR -> 3 LHS_VECTOR

STOP
```

12.6.8 References

[1] Stanley, G. and Swenson, L., *HDB: Object-Oriented Database Utilities for COMET-AR*, NASA Computational Structural Mechanics (CSM) Contract Report, August 1992.

[2] Felippa, Carlos A., *The Computational Structural Mechanics Testbed Architecture: Volume II: Directives*, NASA CR-178385, February 1989.

12.7 Processor SKYs (Direct Linear Equation Solver)

12.7.1 General Description

Processor SKYs is a general purpose, out-of-core, linear equation solver employing the Crout (LDU) factorization of symmetric non-singular matrices stored in skyline storage format (SMT data object class).

SKYs includes special provisions for the particular needs of the AR environment:

- ***Partial factorization.*** Factor only new columns appended to a previously factored matrix (required by the h_s-refinement technique); and
- ***Iterative solution.*** Use of Preconditioned Conjugate Gradient (PCG)[1] iterations using the h_s-refinement block diagonal matrix as a preconditioner (only when h_s-refinement technique is being used).

SKYs employs an optimized two-block factorization algorithm: a factored block, and an updated block, which are dynamically allocated at run time with a size controlled by the user.

Processor SKYs is typically invoked by a high-level AR control procedure, such as AR_CONTROL_1 (via procedure L_STATIC_1), in an adaptive refinement iterations loop.

12.7.2 Command Summary

Processor SKYs follows standard COMET-AR command interface protocol. A summary of SKYs commands is given in Table 12.7-1.

Table 12.7-1 Processor SKYs Command Summary

Command Name	Function	Default
SET BUFFER_SIZE	Specifies size in float words of the memory buffer used by SKYs to store each of the system matrix blocks.	262144 (1 MB single, 2 MB double)
SET CONSTRAINT_SET	Specifies constraint-set number.	1
SET CONV_TOL	Convergence tolerance.	10^{-6}

1. See Section 12.4, *Iterative Linear Equation Solver*, for more details about the PCG algorithm.

Table 12.7-1 Processor SKYs Command Summary (Continued)

Command Name	Function	Default
SET COUPLING	For h_s-refinement, specifies whether coupling load terms between superposed and underlying meshes should be computed using the SIP or can be computed from elements coupling stiffness (stored in EAT).	COMPUTE
SET FIXED_FRAME	Fixed frame flag for h_s-refinement with the iterative solution option.	NO
SET LDIC	Specifies logical device index of computational database.	1
SET LDIE	Specifies logical device index of element matrices database (required only for iterative solutions).	2
SET LDIS	Specifies logical device index of system matrices database.	3
SET LOAD_SET	Specifies load-set number.	1
SET MAX_ITER	Specifies maximum allowable number of iterations.	100
SET MESH	Specifies mesh number.	0
SET STEP	Specifies load step number.	0
SET SOLVER	Specifies type of solver to be used: direct or iterative.	DIRECT
SET UPDATE	For h_s-refinement, specifies whether the load terms of an underlying mesh should be updated using the SIP to account for geometry changes between the superposed mesh and the underlying mesh.	UPDATE
FACTOR/*qualifier*	Partially factor the assembled system matrix.	
SOLVE	Obtain a solution vector.	

12.7.3 Command Definitions

12.7.3.1 FACTOR Command

This is the "go" command for factorization stage. It causes SKYs to factor the assembled matrix using Crout LDU factorization.

Command syntax:

FACTOR/*qualifier*

where

Parameter	Description		
qualifier	Depends on the SOLVER selected (default value: FULL)		
		Direct	**Iterative**
	FULL	full factorization	factor all diagonal mesh blocks
	PARTIAL	only factor new columns	factor diagonal block for last mesh only

12.7.3.2 SOLVE Command

This is the "go" command for solution stage. It causes SKYs to compute the solution vector.

Command syntax:

SOLVE

12.7.3.3 SET BUFFER_SIZE Command

This command defines the size of the memory buffer used by SKYs to hold each assembled system matrix columns block. A single block is required during the solution (SOLVE) stage and two blocks are required during the factorization (FACTOR) stage.

Command syntax:

SET BUFFER_SIZE = *size*

where

Parameter	Description
size	Matrix block size (in floating precision words) (Default 262144)

12.7.3.4 SET CONSTRAINT_SET Command

Defines the constraint set number associated with the element, nodal, and system data. This number should appear as the second cycle number in names of all datasets.

Command syntax:

SET CONSTRAINT_SET = *conset*

where

Parameter	Description
conset	Constraint set number (default value: 1)

12.7.3.5 SET CONV_TOL Command

Defines the convergence tolerance for the PCG iterations with respect to energy error.

Command syntax:

SET CONV_TOL = *contol*

where

Parameter	Description
contol	Convergence tolerance (default value: 10^{-6})

12.7.3.6 SET COUPLING Command

Defines the source for the coupling load terms during iterative h_s-refinement solution.

Command syntax:

SET COUPLING = *source*

where

Parameter	Description
source	The source for the coupling load terms: **LOAD**—read coupling stiffness from EAT (*EltNam*.COUP_STIFF...*mesh* dataset) and multiply by the displacement vector **COMPUTE**—use the SIP to interpolate loads from superposed element to their underlying ancestor elements (default value: COMPUTE)

12.7.3.7 SET FIXED_FRAME Command

Sets the fixed-frame flag to the appropriate state for h_s-refinement solution. SKYs computes the residual vectors by using the "S-Interpolation Property" (SIP) to account for the coupling terms between the superposed mesh and the underlying mesh. This process involves interpolation of load vectors which are needed to be transformed into a global/fixed coordinate system before and after the interpolation, unless a fixed computational frame is used throughout the mesh.

Command syntax:

SET FIXED_FRAME = *flag*

where

Parameter	Description
flag	Fixed computational frame flag (default value: NO)

12.7.3.8 SET LDIC Command

Defines the logical device index for the computational database.

Command syntax:

SET LDIC = *ldic*

where

Parameter	Description
ldic	Logical device index (default value: 1)

12.7.3.9 SET LDIE Command

Defines the logical device index for the element matrices database.

Command syntax:

SET LDIE = *ldie*

where

Parameter	Description
ldie	Logical device index (default value: 2)

12.7.3.10 SET LDIS Command

Defines the logical device index for the system matrices database.

Command syntax:

SET LDIS = *ldis*

where

Parameter	Description
ldis	Logical device index (default value: 3)

12.7.3.11 SET LOAD_SET Command

Defines the load set number associated with the element, nodal, and system data. This number should appear as the first cycle number in names of all datasets.

Command syntax:

SET LOAD_SET = *ldset*

where

Parameter	Description
ldset	Load set number (default value: 1)

12.7.3.12 SET MAX_ITER Command

Defines the maximum allowable number of PCG iterations.

Command syntax:

SET MAX_ITER = *maxiter*

where

Parameter	Description
maxiter	Maximum number of iterations (default value: 100)

12.7.3.13 SET MESH Command

Defines the mesh number for the system equations to be processed. This number should appear as the third cycle number in names of all datasets.

Command syntax:

SET MESH = *mesh*

where

Keyword	Description
mesh	Mesh to be solved (default value: 0)

12.7.3.14 SET SOLVER Command

Defines the solver to be used during h_s-refinement solution.

Command syntax:

SET SOLVER = *solver*

where

Parameter	Description
solver	Type of solver to be used in h_s-refinement solution: DIRECT or ITERATIVE (default value: DIRECT)

12.7.3.15 SET STEP Command

Defines the solution step number (for nonlinear analysis only) associated with the element, nodal, and system data. This number, if defined, should appear as the first cycle number in names of all datasets.

Command syntax:

SET STEP = *step*

where

Parameter	Description
step	Solution step number (default value: 0)

12.7.3.16 SET UPDATE Command

This command sets the update mode for underlying mesh elements during iterative h_s-refinement solution. SKYs is capable of updating the load terms of an underlying mesh using the SIP to account for geometry changes between the superposed mesh and the underlying mesh.

Command syntax:

SET UPDATE = *flag*

where

Parameter	Description
flag	Update load mode: YES/NO (default value: YES)

12.7.4 Database Input/Output

12.7.4.1 Input Datasets

A summary of input datasets required by Processor SKYs is given in Table 12.7-2.

Table 12.7-2 Processor SKYs Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset
NODAL.DOF.*conset.mesh*	NDT	Nodal DOF dataset
EltNam.DEFINITION...*mesh*	EDT	Element definition dataset
EltNam.REFINEMENT...*mesh*	ERT	Element refinement dataset
EltNam.COUP_STIFF...*mesh*	EAT	Element coupling matrix dataset (in file LDIE)
EltNam.MATL_STIFFNESS...*mesh*	EMT	Element matrix dataset (in file LDIE)
SYSTEM.VECTOR.*ldset..mesh*	SVT	System load vector
COLUMN.HEIGHT...*mesh*	SMT	Columns heights (in file LDIS)
DIAGONAL.ADRESSES...*mesh*	SMT	Diagonal elements pointers (in file LDIS)
D.VECTOR...*mesh*	SMT	D vector of Crout LDU decomposition (in file LDIs). This dataset is an input dataset in the SOLVE stage and an output dataset in the FACTOR stage.
STRUCTURE.MATL_STIFFNESS...*mesh*	SMT	The assembled matrix (in file LDIS): unfactored matrix as input to the FACTOR stage and the decomposed matrix as input to the SOLVE stage

12.7.4.2 Output Datasets

A summary of output datasets created by Processor SKYs is given in Table 12.7-3.

Table 12.7-3 Processor SKYs Output Datasets

Dataset	Class	Contents
SYSTEM.VECTOR.*ldset..mesh*	SVT	System load vector
D.VECTOR...*mesh**	SMT	D vector of Crout LDU decomposition (in file LDIS)
STRUCTURE.MATL_STIFFNESS...*mesh*	SMT	The factored system matrix (in file LDIS)

*—created record

12.7.5 Limitations

12.7.5.1 SMT Data Structures Form

SKYs currently recognizes only SMT type data structure as created by ASMs. Other assembly processors within COMET-AR output different types of data structures (such as COMPAXX format) which can not be accessed by SKYs.

12.7.6 Error Messages

SKYs contains extensive error checking. Most of the error messages printed by SKYs are self-explanatory and aim to help the user correct mistakes. Some of the errors may occur at code levels below SKYs (e.g., HDB, DB, GAL, etc.) and SKYs describes those errors to the best of its ability.

The following summarizes error messages related to user interface problems produced by SKYs.

Index	Error Message	Cause	Recommended User Action
1	Unknown SET *variable name* encountered in SKYs.	SKYs user interface cover encountered unrecognized SET variable name.	Check spelling of set *variable name* in CLIP procedure.
2	Unknown *command* encountered in SKYs.	SKYs user interface cover encountered unrecognized command.	Check spelling of *command* in CLIP procedure.
3	Old/new *dataset name* could not be opened in *routine name*.	SKYs could not open a certain dataset.	1. Check execution log for error produced by processors prior to SKYs execution. 2. Verify the particular *dataset name* using the HDBprt processor. 3. Make sure that all required input datasets are present in the database file.

Index	Error Message	Cause	Recommended User Action
5	*Dataset name* could not be closed in *routine name.*	SKYs could not close a certain dataset.	1. Check execution log file; look for errors previously produced by processor SKYs. 2. Verify SKYs is the only processor accessing the database file. (Is ARGx being used in the same directory?)
6	*Dataset name* access problem in *routine name* —or— could not get/put/add/update/ *attribute name* to *dataset name* in *routine name.*	SKYs could not get/put an attribute from/to the dataset name table.	Verify that the particular dataset contains attributes required by SKYs.
7	Could not allocate memory for *array name* in *routine name*	Malloc memory allocation problem encountered in DBmem level	Check with your system manager. You may need permission to access more memory.
8	Zero or negative diagonal term in FACTOR, Equation number = *eqn.*	Assembled matrix is non-positive-definite matrix.	Check your model element connectivity and boundary condition. ARGx may be useful in this regard.

In addition to the above generic messages, SKYs will print any relevant information regarding the problem to assist the user in correcting the error. A full trace-back printout of error messages will follow the first message, and SKYs will attempt to terminate its execution as cleanly as possible (closing opened datasets, releasing memory allocations, etc.).

12.7.7 Examples and Usage Guidelines

12.7.7.1 Example 1: Basic Operation

```
RUN SKYs
                SET MESH        = 1
                SET SOLVER      = DIRECT
                FACTOR/FULL
                SOLVE
STOP
```

In this example, the assembled matrix of mesh 1 is factored out-of-core using the default memory allocation size (e.g., 524288 words) followed by standard forward reduction/back substitution solution for mesh 1 displacement field.

12.7.7.2 Example 2: Iterative Solution (h_s-refinement only)

```
RUN SKYs
                SET MESH            = 1
                SET SOLVER          = ITERATIVE
                SET COUPLING        = LOAD
                SET CONV_TOL        = 10-7
                SET MAX_ITER        = 1000
                FACTOR/PARTIAL
                SOLVE
STOP
```

In this example, the assembled block diagonal matrix of mesh 1 h_s-refinement is factored followed by an iterative solution for the mesh 1 displacement field using 10^{-7} as the solution energy error norm tolerance, allowing up to 1000 iterations, and computing the residual vectors using re-computed element stiffness coupling matrices stored in an EAT dataset.

12.7.8 References

None.

12.8 Processor VEC (Vector Algebra Utility)

12.8.1 General Description

Processor VEC performs a number of algebraic functions using nodally-oriented system vectors and rotation vectors, which are stored in the database as Nodal Vector Table (NVT) and Nodal Attribute Table (NAT) data objects (Reference [1]). Basic facilities exist in VEC for creating, interrogating, and modifying NVT and NAT data objects. VEC is primarily used to write solution procedures to solve specific analysis tasks. An example of a solution procedure developed using VEC is NL_STATIC_1, which uses an arclength-controlled version of a modified Newton-Raphson incremental solution algorithm. VEC is also used in stand-alone mode to pre- or post-process NVT and NAT data objects.

VEC also has the capability to create and modify a Nodal Degree-of-Freedom Table (NDT) data object. This feature is used in certain nonlinear solution procedures to modify the active degrees-of-freedom at a node. Processor VEC should not be used instead of processor COP (specifically designed to define a Nodal DOF Table), but to augment COP's capabilities.

12.8.2 Processor Command Summary

VEC input is governed by the COMET-AR command language CLAMP (Reference [2]); therefore, VEC accepts both CLIP directives [3] and VEC-specific commands. All VEC output are data objects which conform to the object-oriented High-level Database (HDB) data structure formats [1]. Main memory data management is controlled by DB-MEM [4] and global data management is organized by GAL-DBM [5].

The data type of a new object introduced to VEC is specified by the user in an initialization command. The data type of an existing object (residing on a GAL library) is known to VEC, since HDB data objects are self-descriptive. The only data object classes recognized by VEC are NAT, NDT, and NVT. If VEC does create a new data object, its numerical precision is determined by VEC's default floating-point precision, selected when the processor is compiled.

Use of HDB data structures requires that data objects be linked to a Complete Summary of the Model (CSM) data object. In VEC, the requirement of associating a data object with its CSM object is transparent to the user. VEC automatically searches the default GAL library (via default logical device index *ldi*) for the CSM.SUMMARY...*mesh* dataset when the first VEC command is issued. (The default CSM data object can be changed by resetting default *ldi* and *mesh* values using the SET LDI or SET MESH commands.)

Once a vector object has been opened in VEC, it remains open and active unless an explicit CLOSE command is issued. This convention is established to improve computational efficiency.

Since there is no separate object class for rotation pseudo-vectors, a NAT object is used to store this vector type. As a result, some VEC commands have been updated to handle NAT objects as well as NVT objects.

Table 12.8-1 provides a summary of all VEC commands.

Table 12.8-1 VEC Command Summary

Command Name	Function
CLOSE	Explicitly close and save a data object
COMBINE	Compute the linear combination of two vectors
COMPONENT	Select a specific component from a vector
DOT	Compute the inner product of two vectors
DIAG_INV	Computes the inverse of each component of a NVT vector.
FIX	Change the constraint status of a specified freedom
INIT_NAT	Initialize a rotations pseudo-vector (NAT data object)
INIT_NDT	Initialize a Nodal DOF Table (NDT data object)
INIT_NVT	Initialize a Nodal Vector Table (NVT data object)
INIT_VEC	Initialize a Nodal Vector Table (NVT data object)
NORM	Compute the Euclidean or maximum vector norm
PRINT	Print or display the contents of a vector
PROD	Multiply a diagonal matrix times a vector
ROTATE	Update a nodal rotation pseudo-vector
SET	Set default names and parameters
STOP	Exit the VEC processor

12.8.3 Command Glossary

The general form of a VEC algebraic command is symbolically represented as:

Command [/*qualifier*] α **a** * β **b** -> γ **c**

or:

Command [/*qualifier*] γ **c** <- α **a** $\pm$ β **b**

The command items are defined below.

Item	Item Description
a, b	Input vector dataset identifiers, each comprising a logical device index (*ldi*) and dataset name (*dsn*)
c	Output vector dataset identifier, comprising a logical device index (*ldi*) and a dataset name (*dsn*)
α, β, γ	Scalar constants associated with vectors **a**, **b**, and **c**, usually constants or macrosymbol expressions.

Vector dataset identifiers interpreted by VEC are composed of a complete dataset name specification. The format for specifying a vector dataset identifier is as follows.

Vector Dataset Identifier	
[ldi] dsn	
Parameter	**Description**
ldi	GAL library logical device index containing the dataset named *dsn*
dsn	Name of dataset containing the vector object

The *ldi* value in a vector dataset identifier is optional. If the *ldi* value is omitted, the default *ldi* value supplied by the user on a previous SET LDI command is used. The dataset name *dsn*, however, is mandatory and must include all necessary component names and cycle numbers. An actual VEC command would look (generically) like this:

```
Command [/qualifier]  cA [ldiA] dsnA  + cB [ldiB] dsnB  ->  [ldiC] dsnC
```

where *Command* is a VEC command and *qualifier* is an optional command qualifier; *cA* and *cB* are constants; *dsnA*, *dsnB*, and *dsnC* are dataset names; and *ldiA*, *ldiB*, and *ldiC* are GAL library logical device indices. Using the CLIP directive, *DEFINE, the user can rewrite this command using a math-like syntax that resembles the symbolic representation. The following formal statements illustrate the use of *DEFINE directives used in conjunction with a generic VEC command.

```
*def/d a = cA
*def/d b = cB
*def/a VectorA = [ldiA]  dsnA
*def/a VectorB = [ldiB]  dsnB
*def/a VectorC = [ldiC]  dsnC
Command [/qualifier]  <a> <VectorA>  +  <b> <VectorB>  ->  <VectorC>
```

12.8.3.1 CLOSE Command

The CLOSE command explicitly saves and closes a data object currently open in VEC. This function is automatically performed when the VEC processor is terminated with a STOP or RUN command. By explicitly closing an object, however, the user can control VEC's use of main memory.

```
CLOSE [a]
```

The command items are defined below.

Item	Item Description
a	Input dataset identifier: [*ldi*] *dsn*

If no dataset identifier is specified, then all open data objects are saved and closed.

12.8.3.2 COMBINE Command

The COMBINE command computes the linear combination of up to two vectors. Each vector may be scaled by a constant. The use of this command is not limited to combining two vectors. By using only one dataset identifier this command can be used to perform a range of functions such as clearing, initializing, scaling, and copying a vector. This command applies to NVT data objects only, except when a vector is scaled it accepts NVT or NAT (pseudo-vector) data objects.

COMBINE **c** <- [α] **a** [± [β] **b**]

The command items are defined below.

Item	Item Description
a, b	Input dataset identifiers: [*ldi*] *dsn*
c	Output dataset identifier: [*ldi*] *dsn*
α, β	Constants multiplying vectors **a** and **b** (default values: 1.0)

12.8.3.3 COMPONENT Command

The COMPONENT command either extracts a specified component from a vector or replaces a vector component with a new value. The direction of the arrow operator indicates which function to perform. The extracted component value is assigned to a designated macrosymbol and is also printed in the output file. This command is only valid for NVT data objects.

To extract a specific component from a vector object use the following command.

COMPONENT **a** { *i* | *node, dof* } -> γ

The command items are defined below.

Item	Item Description
a	Input dataset identifier: [ldi] dsn
i	Vector component index: $1 \le i \le$ length(**a**)
node, dof	Node and DOF index corresponding to a component: $1 \le node \le nnode$ and $1 \le dof \le ndof$

Item	Item Description
γ	Name of macrosymbol to receive the vector component value

To change the value of a specified component in a vector dataset, use the following command.

COMPONENT **a** { *i* | *node, dof* } <- γ

The command items are defined below.

Item	Item Description
a	Output dataset identifier: [*ldi*] *dsn*
i	Vector component index: 1 ≤ *i* ≤ length(**a**)
node, dof	Node and DOF index corresponding to a component: 1 ≤ *node* ≤ *nnode* and 1 ≤ *dof* ≤ *ndof*
γ	Scalar constant assigned to the component specified by *i* or *node,dof*

12.8.3.4 DIAG_INV Command

The DIAG_INV command computes the inverse of Nodal Vector Table (NVT) data object.

DIAG_INV **a** -> **c**

The command items are defined below.

Item	Item Description
a	Input dataset identifiers: [*ldi*] *dsn*
c	Output dataset identifier: [*ldi*] *dsn*

12.8.3.5 DOT Command

The DOT command computes an inner vector product and is only valid for NVT data objects.

DOT **a** * **b** -> γ

The command items are defined below.

Item	Item Description
a, b	Input dataset identifiers: [*ldi*] *dsn*

Item	Item Description
γ	Name of macrosymbol to receive the dot product value

12.8.3.6 FIX Command

The FIX command assigns the state attribute of a component in an NDT data object to be zero (i.e., a suppressed DOF).

FIX *node, dof* **d**

The command items are defined as follows:

Item	Item Description
d	Input/output NDT dataset identifier: [*ldi*] *dsn*
node, dof	Node and DOF index corresponding to the suppressed freedom: $1 \le node \le nnode$ and $1 \le dof \le ndof$

12.8.3.7 INIT_NAT Command

The INIT_NAT command creates and initializes new rotation pseudo-vectors stored in a Nodal Attribute Table (NAT) data object.

INIT_NAT ***a*** [*nnode, ndof*]

The command items are defined below.

Item	Item Description
a	Output NAT dataset identifier: [*ldi*] *dsn*
nnode	Number of nodes (columns) in the table. If omitted, the attribute value of *Nnode* in the CSM data object is used.
ndof	Number of degrees-of-freedom per node (default value: 3)

12.8.3.8 INIT_NDT Command

The INIT_NDT command creates and initializes a new Nodal DOF Table (NDT) data object. The attributes associated with each node are initialized so that all nodes are active and all nodal degrees-of-freedom are FREE. The components are numbered consecutively.

INIT_NDT **d** [*nnode, ndof*]

The command items are defined below.

Item	Item Description
d	Output NDT dataset identifier: [*ldi*] *dsn*
nnode	Number of nodes (columns) in the table. If omitted, the attribute value of *Nnode* in the CSM data object is used.
ndof	Number of degrees-of-freedom per node. If omitted, the value of *Ndofn* in the CSM data object is used. If the CSM isn't open, *ndof*=6 is used.

12.8.3.9 INIT_NVT Command

The INIT_NVT command creates and initializes a new Nodal Vector Table (NVT) data object.

INIT_NVT **a** [*nnode, ndof*]

The command items are defined below.

Item	Item Description
a	Output NVT dataset identifier: [*ldi*] *dsn*
nnode	Number of nodes (columns) in the table. If omitted, the attribute value of *Nnode* in the CSM data object is used.
ndof	Number of degrees-of-freedom per node (default value: 3)

12.8.3.10 INIT_VEC Command

This command is a synonym for the INIT_NVT command. See the description of INIT_NVT for usage details.

12.8.3.11 NORM Command

The NORM command either computes the Euclidean norm or the maximum norm of a vector, and is only valid for NVT data objects. The command for the Euclidean vector norm takes the following form:.

NORM **a** -> γ

The command items are defined below.

Item	Item Description
a	Input dataset identifier: [*ldi*] *dsn*

Item	Item Description
γ	Name of macrosymbol to receive the Euclidean vector norm value

The command for the maximum vector norm takes the following form:

```
NORM /MAX   a -> γ { i | node, dof }
```

where the command items are defined as follows:

Item	Item Description
a	Input dataset identifier: [*ldi*] *dsn*
γ	Name of macrosymbol to receive the maximum vector norm value
i	Name of macrosymbol to receive the maximum-component index value
node, dof	Names of macrosymbols to receive the maximum-component (*node, dof*) values

12.8.3.12 PRINT Command

The PRINT command displays the contents of an NVT or an NAT data object. Print output can be redirected to a file by changing the default output unit number using the SET OUTPUT_UNIT command described in this Command Glossary. The name of the output is set with the SET FILENAME command, also described in the Command Glossary.

```
PRINT   a
```

The command items are defined below.

Item	Item Description
a	Input dataset identifier: [*ldi*] *dsn*

12.8.3.13 PROD Command

The PROD command multiplies a diagonal matrix times a vector and is only valid for NVT data objects.

```
PROD   a * b  ->  c
```

The command items are defined below.

Item	Item Description
a, b	Input NVT dataset identifiers: [*ldi*] *dsn*
c	Output NVT dataset identifier: [*ldi*] *dsn*

12.8.3.14 ROTATE Command

The ROTATE command updates nodal rotation triads (pseudo-vectors), stored in an NAT data object.

ROTATE **a** * β **b** -> **c**

The command items are defined below.

Item	Item Description
a	Input (old) NAT rotation pseudo-vector dataset identifier: [*ldi*] *dsn*
b	Input NVT dataset identifier: [*ldi*] *dsn*
c	Output (updated) NAT rotation pseudo-vector dataset identifier: [*ldi*] *dsn*

12.8.3.15 SET Command

The SET command is used to specify name and parameter values.

SET *keyword* = *value*

The command items are defined below.

Keyword	Default	Item Description
DOFN	d1, d2, d3, theta1, theta2, theta3	Nodal DOF name list
MESH	0	Mesh number
NDOFN	6	Number of nodal DOFs (must agree with no. of items in DOFN list)
LDI	1	GAL library logical device index value
OUTPUT_UNIT	6	FORTRAN output logical unit number
STEP	0	Solution step number

12.8.3.16 STOP Command

The STOP command saves and closes all open VEC data objects and database files and returns control to the parent processor. The COMET-AR command "RUN *processor_name*" will perform this same function in addition to transferring control to processor named *processor_name*.

STOP

12.8.4 Database Input/Output Summary

12.8.4.1 Input Datasets

VEC input, depending on the selected command, consists of data objects with generic dataset names listed in Table 12.8-2. The data object types required by each specific command are discussed in Section 12.8.2, *Processor Command Summary*.

A CSM object (typically contained in a dataset named CSM.SUMARY...*mesh*) is associated with each data object recognized by VEC. Explicit specification of the CSM object associated with every data object is not required, VEC locates the CSM object on the database using the LDI value and mesh number specified with the SET LDI and SET MESH commands. If the LDI value or mesh number is not explicitly set, then VEC uses its internal values defined when the processor starts. If a CSM object is not found, then one will be created using either the user-supplied LDI value and mesh number or VEC's default LDI value and mesh number.

Table 12.8-2 Processor VEC Input Datasets

Dataset	Class	Description
CSM.SUMMARY...*mesh*	CSM	Complete Summary of the Model (CSM data object)
NODAL.*AttName.step..mesh*	NAT	Nodal Attribute Table (NAT) of pseudo-vectors used to update nodal rotation triads. The name of an NAT data object is completely specified by the user, including component names and cycle numbers.
NODAL.DOF.*conset.mesh*	NDT	Nodal DOF Table (NDT). The name of an NDT data object is completely specified by the user, including component names and cycle numbers
NODAL.*VecName.step..mesh*	NVT	Nodal Vector Table (NVT). The name of an NVT data object is completely specified by the user, including component names and cycle numbers.

12.8.4.2 Output Datasets

VEC output, depending on the selected command, is a data object with a generic dataset name listed in Table 12.8-3. The data object types required by each specific command are discussed in Section 12.8.2, *Processor Command Summary*.

A CSM object (typically contained in a dataset CSM.SUMARY...*mesh*) is associated with each data object recognized by VEC. Explicit specification of the CSM object associated with every data object are not required, VEC locates the CSM object on the database using the LDI value and mesh number specified with the SET LDI and SET MESH commands. If the LDI value or mesh number is not explicitly set, then VEC uses its internal values which are defined when the processor starts. If a CSM object is not found, then one will be created using either the user-supplied LDI value and mesh number or VEC's default LDI value and mesh number. VEC initializes a new CSM object's nodal attributes with the values of the SET command parameters NDOFN and DOFN.

If VEC creates a new data object, its numerical precision is determined by VEC's default floating-point precision, selected when the processor is compiled.

Table 12.8-3 Processor VEC Output Datasets

Dataset	Class	Description
CSM.SUMMARY...*mesh*	CSM	Complete Summary of the Model (CSM data object)
NODAL.*AttName.step..mesh*	NAT	Nodal Attribute Table (NAT) of pseudo-vectors used to update nodal rotation triads. The name of an NAT data object is completely specified by the user, including component names and cycle numbers.
NODAL.DOF.*conset.mesh*	NDT	Nodal DOF Table (NDT). The name of an NDT data object is completely specified by the user, including component names and cycle numbers.
NODAL.*VecName.step..mesh*	NVT	Nodal Vector Table (NVT). The name of an NVT data object is completely specified by the user, including component names and cycle numbers.

12.8.5 Limitations

Processor VEC has no inherent limitations.

12.8.6 Error Messages

VEC produces user-friendly error messages. The messages often include complete descriptions of all vectors involved (i.e., *ldi* and *dataset_name*) with a comment about what to do next.

12.8.7 Examples and Usage Guidelines

The example script shown in Table 12.8-4 is located in the VEC master source directory ($AR_VEC) and is used to test basic VEC functionality. It contains instructive comments and illustrates the syntax for almost every VEC command. Study this file before applying VEC to specific analysis tasks.

Table 12.8-4 Example 1: Test Script for VEC

```
*sys rm VEC_TEST.DBC
*open VEC_TEST.DBC

*def/i vec_size == 10
```

Table 12.8-4 Example 1: Test Script for VEC (Continued)

```
*def/i vec_io   == 6

set output_unit = <vec_io>

*remark ================================
*remark TESTING OPERATIONS on NVT OBJECT
*remark ================================
*remark
*remark
*remark Initialize vector A : VECCLR  VECINVT
*remark ====================================

  init_vec 1 a.a <vec_size>
  1 a.a <- 1.0
  print 1 a.a

*remark
*remark Initialize vector B : VECCLR  VECINVT
*remark ====================================

  init_vec 1 b.b <vec_size>
  1 b.b <- 2.0
  print 1 b.b

*remark
*remark Linear combination of input vectors C = 2.0*A + B : VECADD
*remark ============================================================

  1 c.c <- 2.0 1 a.a + 1 b.b
  print 1 c.c

*remark
*remark Linear combination of input vectors C = 2.0*A + 3.0*A : VECADD
*remark ================================================================

  1 a.a <- 2.0 1 a.a + 3.0 1 a.a
  print 1 a.a

*remark
*remark Scaled copy of vector C = 2.0*C : VECSCL
*remark ========================================

  1 c.c <- 2.0 1 c.c
  print 1 c.c

*remark
*remark Direct copy of vector D = C : VECCOP
*remark ===================================

  d.d <- 1 c.c
  print 1 d.d

*remark
*remark Extract component value from vector: VECCMP
*remark ============================================

  comp 1 b.b 9 3 -> node2
  comp   c.c 9 3 -> node1
  *show macro node1
  *show macro node2

*remark
*remark Set component value of vector: VECCMP
```

Table 12.8-4 Example 1: Test Script for VEC (Continued)

```
*remark ====================================

  comp  1 b.b 9 3 <- <node1>
  print 1 b.b 9 3
  comp    c.c 9 3 <- <node2>
  print 1 c.c 9 3

*remark
*remark Compute DOT product: VECDOT
*remark ===========================

  dot   d.d *   d.d -> scaleD
  dot 1 a.a * 1 d.d -> scaleAB
  *show macro scaleD
  *show macro scaleAB

*remark
*remark Compute vector product (term by term): VECPROD
*remark ============================================

  prod    b.b *   b.b -> e.e
  print   e.e

  prod  1 e.e *   c.c -> 1 f.f
  print 1 f.f

  prod  1 f.f * 1 f.f -> 1 f.f
  print 1 f.f 1
  print 1 f.f 0 4

*remark
*remark Compute norm: VECnrm
*remark ====================

  norm 1 a.a -> normA
  *show macro normA

*remark
*remark Compute MAX norm: VECmax
*remark ========================

  norm/max 1 b.b -> MaxNormB NodeB DofB
  *show macro MaxNormB
  *show macro NodeB
  *show macro DofB

*remark
*remark
*remark ================================
*remark TESTING OPERATIONS on NDT OBJECT
*remark ================================
*remark
*remark
*remark Initialize vector DOF Table : VECINDT
*remark =====================================

  init_ndt 1 n.n <vec_size>

  fix   1 n.n 1 6
  fix   1 n.n 1 5

  fix   1 n.n 1 4
```

Table 12.8-4 Example 1: Test Script for VEC (Continued)

```
  free  1 n.n 1 5

*remark
*remark
*remark ================================
*remark TESTING OPERATIONS on NAT OBJECT
*remark ================================
*remark
*remark
*remark Initialize NAT vectors A : VECINAT
*remark ============================

  init_nat 1 q.q <vec_size>
  init_nat 1 s.s <vec_size> 3
  print/nat 1 s.s

*remark Testing rotation Pseudo_vector Update:VECSPN
*remark ==========================================

  rotate 1 s.s * 1.0 1 a.a -> 1 s.s
  print/nat 1 s.s

  rotate 1 s.s * 0.2 1 a.a -> 1 t.t
  print/nat   t.t

  1 u.u <- 1 t.t
  print/nat  u.u

*remark
*remark
*remark Closing all open HDB/DB objects
*remark ==============================

  close
```

12.8.8 References

[1] Stanley, G., and Swenson, L., *HDB: Object-Oriented Database Utilities for COMET-AR*, NASA Computational Structural Mechanics (CSM) Contract Report, August 1992.

[2] Felippa, C., *A Command Language for Applied Mechanics Processors, Volume I: The Language*, NASA CR-178384, 1988.

[3] Felippa, C., *A Command Language for Applied Mechanics Processors Volume II: Directives*, NASA CR-178385, 1989.

[4] Stehlin, B., *DB/MEM: Generic Database Utilities for the COMET-AR Testbed*, NASA Computational Structural Mechanics (CSM) Contract Report, May 1992.

[5] Felippa, C., Regelbrugge, M., and Wright, M., *The Computational Structural Mechanics Testbed Architecture, Volume IV: The Global Database Manager GAL-DBM*, NASA CR-178387, 1989.

12.9 Processor VSS (Vectorized Sparse Solver)

12.9.1 General Description

Processor VSS is a very fast direct linear equation solver developed at NASA that operates on sparse matrices and employs optimal equation renumbering to minimize the number of floating point operations.

12.9.2 Command Summary

Processor ITER follows standard COMET-AR command interface protocol. A summary of ITER commands is given below in Table 12.9-1.

Table 12.9-1 Processor VSS Command Summary

Command Name	Function	Default Value
SET CONSTRAINT_SET	Specifies constraint-set number	1
SET LDIC	Specifies logical device index of computational database	1
SET LDIS	Specifies logical device index of system database	3
SET LOAD_SET	Specifies load set number	1
SET MESH	Specifies mesh number	0
SET STEP	Specifies load step number	0
SOLVE	Obtain a solution using PCG iterations	

12.9.3 Command Definitions

12.9.3.1 SOLVE Command

This is the "go" command for processor VSS. It causes VSS to both factor the assembled matrix and solve for the solution vector via forward and back substitution.

Command syntax:

```
SOLVE
```

12.9.3.2 SET CONSTRAINT_SET Command

This command defines the constraint set number associated with the element, nodal, and system data. This number should appear as the second cycle number in names of all datasets.

Command syntax:

SET CONSTRAINT_SET = *conset*

where

Parameter	Description
conset	Constraint set number. (default value: 1)

12.9.3.3 SET LDIC Command

This command defines the logical device index for the computational database.

Command syntax:

SET LDIC = *ldic*

where

Parameter	Description
ldic	Logical device index. (default value: 1)

12.9.3.4 SET LDIS Command

This command defines the logical device index for the system database.

Command syntax:

SET LDIS = *ldis*

where

Parameter	Description
ldis	Logical device index. (default value: 3)

12.9.3.5 SET LOAD_SET Command

This command defines the constraint set number associated with the element, nodal, and system data. This number should appear as the first cycle number in names of all datasets.

Command syntax:

SET LOAD_SET = *ldset*

where

Parameter	Description
ldset	Load set number (default value: 1)

12.9.3.6 SET MESH Command

This command defines the mesh number for the system equations to be processed. This number should appear as the third cycle number in names of all datasets.

Command syntax:

SET MESH = *mesh*

where

Parameter	Description
mesh	Mesh to be solved (default value: 0)

12.9.3.7 SET STEP Command

This command defines the solution step number (for nonlinear analysis only) associated with the element, nodal, and system data. This number, if defined, should appear as the first cycle number in names of all datasets.

Command syntax:

SET STEP = *step*

where

Parameter	Description
step	Solution step number (default value: 0)

12.9.4 Database Input/Output

12.9.4.1 Input Datasets

A summary of input datasets required by Processor VSS is given below in Table 12.9-2.

Table 12.9-2 Processor VSS Input Datasets

Dataset	Class	Contents	
CSM.SUMMARY...*mesh*	CSM	Model summary dataset	
SYSTEM.VECTOR.*ldset..mesh*	SVT	System load vector (right-hand-side of equation system).	
STRUCTURE.MATL_STIFFNESS...*mesh*	System Matrix	Assembled system matrix (in file LDIS). Records description:	
		Record Name	**Description**
		COLLTH	Columns (rows) heights
		COLPTR	Diagonal elements pointers
		ROWS	Identity of non-zero elements in each row
		DIAG	Diagonal terms
		COEFS	Off-diagonal non-zero terms

12.9.4.2 Output Datasets

A summary of output datasets created by Processor VSS is given below in Table 12.9-3.

Table 12.9-3 Processor VSS Output Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary table
SYSTEM.VECTOR.*ldset..mesh**	SVT	System solution vector

12.9.5 Limitations

12.9.5.1 Memory Limitation

VSS is currently limited to in-core operation (i.e., the assembled and factored matrices must fit completely in memory). Due to the optimal renumbering scheme this does not usually cause difficulties for structural problems until the problem size is of the order of 10^5 equations.

12.9.5.2 Multiple Right-Hand-Sides

VSS currently does not save (i.e., output) the factored matrix. It repeats the matrix factorization every time the SOLVE command is issued, even if only the right-hand-side has changed.

12.9.6 Error Messages

To be documented by the developer(s) of VSS.

12.9.7 Examples and Usage Guidelines

12.9.7.1 Example 1: Basic Operation of VSS

```
RUN VSS
                SET MESH          = 1
                SOLVE
STOP
```

In this example, the assembled matrix of mesh 1 is factored followed by an iterative solution for mesh 1 displacement field using 10^{-7} as the solution energy error norm tolerance and allowing up to 1000 iterations.

12.9.8 References

None.

13 Special-Purpose Processors

13.1 Overview

In this chapter we describe special-purpose COMET-AR processors that do not fit in any of the other categories. A summary of currently available special-purpose processors is given in Table 13.1-1.

Table 13.1-1 Outline of Chapter 13: Special-Purpose Processors

Section	Processor	Function
13.2	AMPC	Automatic multi-point constraint (MPC) generator for suppression of spurious shell drilling rotational DOFs.
13.3	COMET-AR	COMET-AR macro-processor used to start-up the system (may contain some or all of the other COMET-AR processors as internal processors).
13.4	TRIAD	Generates new computational reference frames (triads) at nodes that require stabilization of drilling DOFs in response to the AUTO_TRIAD option provided by most COMET-AR Solution Procedures.

13.2 AMPC (Automatic Multipoint Constraint)

13.2.1 General Description

Processor AMPC is used to suppress shell-element drilling rotational freedoms selectively by applying multipoint constraints to the nodal rotations in the computational frame. The multipoint constraints generated by the AMPC processor are stored in the NODAL.DOF dataset and are consistent with any other multipoint constraints that may exist in the model (e.g., due to boundary conditions and/or due to constraint-based refinement). Processor AMPC examines the NODE.DRIL dataset which identifies nodes that require drilling stabilization, and for those nodes generates a multipoint constraint equation defined as follows:

$$\theta^c_{max} \bullet n^c_{max} \geq \theta^c_{minmax} \bullet n^c_{minmax} \geq \theta^c_{min} \bullet n^c_{min}$$

$$\theta^c_{max} = \left(\frac{-1}{n^c_{max}}\right)(n^c_{minmax} \bullet \theta^c_{minmax} + n^c_{min} \bullet \theta^c_{min})$$

where θ^c_i denotes the i^{th} computational rotation freedom at the node and **n** is the element-averaged nodal normal (defined in the NODE.NORM dataset). The subscripts *max, minmax, and min* denote the rotation DOFs which are the most-closely, intermediately, and least-closely aligned with the nodal normal.

Processor AMPC is automatically invoked via the AUTO_MPC solution procedure option (for example, see the L_STATIC procedure in the COMET Procedure Manual).

13.2.2 Command Summary

Processor AMPC follows standard COMET-AR command interface protocol. A summary of recognized commands is given in Table 13.2-1.

Table 13.2-1 Processor AMPC Command Summary

Command Name	Function	Default Value
SET CONSTRAINT_SET	Specifies constraint-set number	1
SET LDI	Specifies logical device index of computational database	1
SET LOAD_SET	Specifies load-set number	1
SET MESH	Specifies mesh number to be processed	0
MPC	Generate the MPCs for the specified mesh	
SET STEP	Specifies step number	0

13.2.3 Command Definitions

13.2.3.1 SET CONSTRAINT_SET Command

This command defines the constraint set number associated with the element and nodal data. This number should appear as the second cycle number in names of all element and nodal datasets.

Command syntax:

SET CONSTRAINT_SET = *conset*

where

Parameter	Description
conset	Constraint set number (default value: 1)

13.2.3.2 SET LDI Command

This command defines the logical device index for the computational database.

Command syntax:

SET LDI = *ldi*

where

Parameter	Description
ldi	Logical device index (default value: 1)

13.2.3.3 SET LOAD_SET Command

This command defines the load set number associated with the element and nodal data. This number should appear as the first cycle number in names of all datasets.

Command syntax:

SET LOAD_SET = *loadset*

where

Parameter	Description
loadset	Load set number (default value: 1)

13.2.3.4 SET MESH Command

This command defines the mesh number associated with the element and nodal data. This number should appear as the third cycle number in names of all datasets.

Command syntax:

SET MESH = *mesh*

where

Parameter	Description
mesh	Mesh number (default value: 0)

13.2.3.5 SET STEP Command

This command defines the step number associated with the element and nodal data for nonlinear analyses. This number should appear as the first cycle number in names of all datasets.

Command syntax:

SET STEP = *step*

where

Parameter	Description
step	Step number (default value: 0)

13.2.3.6 MPC Command

This is the "go" command for processor AMPC. It causes AMPC to generate the multipoint constraints for the mesh.

Command syntax:

MPC

13.2.4 Database Input/Output

13.2.4.1 Input Datasets

A summary of input datasets required by Processor AMPC is given in Table 13.2-2.

Table 13.2-2 Processor AMPC Input Datasets

Dataset	Class	Contents
CSM.SUMMARY...*mesh*	CSM	Model summary dataset
NODAL.DOF..*conset.mesh*	NDT	Nodal DOF dataset
NODAL.DRILL_FLAG...*mesh*	NAT	Nodal drilling stabilization flags
NODAL.NORMAL...*mesh*	NAT	Average element nodal normal vectors

13.2.4.2 Output Datasets

A summary of output datasets created by Processor AMPC is given below in Table 13.2-3.

Table 13.2-3 Processor AMPC Output Datasets

Dataset	Class	Contents
NODAL.DOF..*conset.mesh*	NDT	Nodal DOF dataset

13.2.5 Limitations

13.2.5.1 Tolerance Sensitive

AMPC depends on the NODAL.NORMAL and NODAL.DRILL_FLAG datasets. The nodal normals are computed by an ES*i* element processor and are an approximation to the actual shell normals at nodal points (computed as the average normal for all elements connected to the node point). The drilling flags are also set by an ES*i* element processor by comparing each element's normal at each node to the approximated shell normal at the node. If all elements attached to a node have normals that are parallel, within a user-specified tolerance, to the approximated nodal normal, then the drilling flag of that node is set ON. This algorithm is not a robust method for setting the drilling flags and in certain cases, such as junctures in build-up structures, the averaged nodal normal may not be a good approximation to the actual drilling direction. In all cases the algorithm is sensitive to the tolerance specified by the user for setting the drilling flags (a too small tolerance may yield an ill-conditioned stiffness matrix and a too large tolerance may yield an overly constrained matrix).

13.2.6 Error Messages

AMPC contains extensive error checking. Most of the error messages printed by AMPC are self-explanatory and aim to help the user correct mistakes. Some of the errors may occur at code levels below AMPC (e.g., HDB, DB, GAL, etc.) and AMPC describes those errors to the best of its ability.

The following summarizes the error messages related to user interface problems produced by AMPC.

Index	Error Message	Cause	Recommended User Action
1	Unknown SET *parameter name* encountered in AMPC.	AMPC encountered an unrecognized SET parameter name.	Check the spelling of *parameter name* in the CLIP procedure.
2	Unknown *command* encountered in AMPC.	AMPC encountered an unrecognized COMMAND.	Check the spelling of *command* in the CLIP procedure.
3	Old/new *dataset-name* could not be opened in *routine name.*	AMPC could not open the named dataset.	1. Check execution log file for errors produced by processors prior to AMPC execution. 2. Try to verify the particular *dataset-name* using the HDBprt processor. 3. Make sure that all required input datasets are present in the database file.
4	*Dataset-name* could not be closed in routine name.	AMPC could not close the named dataset.	1. Check the execution log file for errors previously produced by processor AMPC. 2. Verify that AMPC is the only processor accessing the database file. (Is ARGx being used in the same directory?).
5	*Dataset-name* access problem encountered in *routine-name* or Could not get/put/add/update *attribute-name* to *dataset name* in *routine-name.*	AMPC could not get/put an attribute from/to the dataset-name table.	Verify that the particular *dataset-name* contain attributes required by AMPC.

In addition to the above generic messages, AMPC will print any relevant information regarding the problem such as element data, nodal data, and geometry information to assist the user in correcting the error. A full trace-back printout of error messages will follow the first message, and AMPC will attempt to terminate its execution as cleanly as possible by closing open datasets, releasing memory allocations, etc.

13.2.7 Examples and Usage Guidelines

13.2.7.1 Example 1: Basic Operation

```
*run AMPC
                    SET MESH                    = 1
                    MPC
stop
```

In this example, multipoint constraints are generated for mesh=1.

13.2.8 References

None.

13.3 Processor COMET-AR (System Macroprocessor)

13.3.1 General Description

Processor COMET-AR is a macroprocessor for the COMET-AR system. It is used to start COMET-AR and remains in the background to run individual COMET-AR processors in response to the RUN command. Processor COMET-AR may physically contain some or all of the other COMET-AR processors, embedded as internal processors, depending on how COMET-AR has been installed on your operating system.

13.3.2 Commands

The COMET-AR macroprocessor recognizes only one command, the RUN command, which has the following format:

RUN *processor_name*

where *processor_name* represents the name of any of the other COMET-AR processors described in this manual. The RUN command executes the named processor, which then responds to its own local commands until a STOP command is issued to that processor; control is then returned to the COMET-AR macroprocessor. To properly terminate the COMET-AR macro-processor, issue a *STOP directive.

13.3.3 Sample Usage

```
comet-ar                          .  start up macro-processor
          RUN processor_name      .  execute processor
                :::               .  processor commands
          STOP                    .  terminate processor
*stop                             .  terminate macro-processor
```

13.4 Processor TRIAD (Computational Frame Realignment)

13.4.1 General Description

Processor TRIAD is used to selectively replace computational nodal triads residing in the QJJT.BTAB dataset with new triads that permit explicit suppression of shell-element drilling rotational freedoms. The nodal triads stored in this dataset represent 3x3 transformation matrices relating the global coordinate system to the computational coordinate system at the node. Each row of the 3x3 matrix (or triad) represents one of the computational unit vectors expressed in the global cartesian basis. Processor TRIAD examines the NODE.DRIL dataset, which identifies nodes that require drilling stabilization, and replaces the original triad with a new triad defined as follows:

$$\mathbf{z}_c = \mathbf{n}$$

$$\mathbf{y}_c = \mathbf{g}$$

$$\mathbf{x}_c = \mathbf{y}_c \times \mathbf{z}_c$$

where $\mathbf{x}_c$, $\mathbf{y}_c$, $\mathbf{z}_c$ denote the computational unit vectors at the node, $\mathbf{n}$ is the element-averaged nodal normal (defined in the NODE.NORM dataset), and $\mathbf{g}$ is the first global-cartesian unit vector that makes an angle of at least 45 degrees with $\mathbf{n}$. Processor TRIAD does not replace the computational triads at nodes for which user boundary conditions have been prescribed for any DOFs. Nodal loads and user-specified multipoint constraints (MPCs) should not be used in conjunction with processor TRIAD (or the AUTO_TRIAD option), as such loads and MPCs are always expressed in the computational directions, and these directions may be inadvertently modified by TRIAD. Processor TRIAD is automatically invoked via the AUTO_TRIAD solution procedure option (e.g., see procedure L_STATIC in the COMET Procedure Manual). If the AUTO_DOF_SUP solution procedure option is also selected, then the θ_c DOF at nodes whose triads have been updated by TRIAD will automatically be suppressed.

13.4.2 Command Summary

Processor TRIAD follows standard COMET-AR command interface protocol. A summary of valid commands is given in Table 13.4-1.

Table 13.4-1 Command Summary for Processor Triad

Command	Function
LDI	Reset database logical device index
ICON	Reset constraint set index
GO	Selectively replace computational triads in QJJT dataset

13.4.3 Command Definitions

13.4.3.1 LDI Command

The LDI command resets the database logical device index from its default value.

Command Format:

LDI = *ldi*

where the integer *ldi* is the logical device index. (Default value: 1)

13.4.3.2 ICON Command

The ICON command resets the constraint set index from its default value.

Command Format:

ICON = *icon*

where *icon* is the constraint set index appearing in the name of dataset NODAL.DOF..*icon*. (Default value: 1)

13.4.3.3 GO Command

This is the action command. It causes the global-to-computational nodal triads in the NODAL.DOF dataset to be selectively replaced by new triads such that the computational z_c axis is parallel to the average element normal at the node.

Command Format:

GO

13.4.4 Database Input/Output

13.4.4.1 Input Datasets

A summary of input datasets required by Processor TRIAD is given in Table 13.4-2.

Table 13.4-2 Processor TRIAD Input Datasets

Dataset	Class	Contents
NODAL.NORMAL	NAT	Average element nodal normal vectors
NODAL.DRILL_FLAG	NAT	Nodal drilling stabilization flags
NODAL.DOF..*icon*	NDT	Nodal DOF table
NODAL.TRANSFORMATION	NTT	Initial nodal computational triads

13.4.4.2 Output Datasets

A summary output datasets created by Processor TRIAD is given in Table 13.4-3.

Table 13.4-3 Processor TRIAD Output Datasets

Dataset	Class	Contents
NODAL.TRANSFORMATION	NTT	Modified nodal computational triads

13.4.5 Limitations

1) Processor TRIAD is not suitable for use in conjunction with multi-point constraints (MPCs), as triad replacements at nodes involved in MPCs may invalidate the user's constraint relations.

2) Processor TRIAD is also not suitable for use with nodal (i.e., concentrated) loads, as such loads are by definition specified in the computational coordinate system at each node and these directions are likely to be altered by triad replacement.

13.4.6 Error Messages

1) "Cannot open NODAL.NORMAL dataset." — The dataset is not present in the currently open database or has been corrupted.

2) "Cannot open NODAL.DRILL_FLAGG dataset." — The dataset is not present in the currently open database or has been corrupted.

3) "Cannot open NODAL.TRANSFORMATION dataset." — The dataset is not present in the currently open database or has been corrupted.

4) "Cannot open Nodal DOF dataset." — The NODAL.DOF..*icon* dataset is not present in the currently open database or has been corrupted.

5) "Cannot find good global axis." — Implies that all three of the global axes are within 45 degrees of the average shell-element normal at a given node. The NODAL.NORMAL dataset was not properly formed, or the code has been corrupted.

6) "Command error." — The user has issued a command to Processor TRIAD that is not contained in the list of valid commands summarized in Table 13.4-1.

13.4.7 Usage Guidelines

1) It is best to employ Processor TRIAD only via the AUTO_TRIAD solution procedure option. This will insure that it is invoked at the right stage in the analysis and is synchronous with other COMET-AR solution procedure options such as AUTO_DOF_SUP.

2) The main purpose of Processor TRIAD is to enable the AUTO_TRIAD solution procedure option, which aligns selected nodal triads (i.e., computational-to-global transformation matrices) with the average shell-element normal vectors so that drilling rotational DOFs can be subsequently suppressed. For this purpose, be sure to select the AUTO_DOF_SUP solution procedure option in addition to the AUTO_TRIAD option, so that the necessary DOF suppressions will be imposed.

3) Do not use Processor TRIAD (or the AUTO_TRIAD option) in conjunction with multipoint constraints (MPCs) unless the MPCs involve only nodes that are not eligible for triad replacement by TRIAD (i.e., nodes at which sufficient drilling rotational stiffness is already present).

4) Do not apply nodal (concentrated) loads or element loads referred to the computational bases at nodes where triads may be replaced by Processor TRIAD. Nodes eligible for triad replacement are those at which insufficient drilling rotational stiffness exists and at which no boundary conditions have been prescribed.

13.4.8 References

None.

14 Post-Processors

14.1 Overview

In this chapter, various post-processors implemented in COMET-AR are described. These processors are used primarily for solution (and model) display. A summary of currently available post-processors is given in Table 14.1-1.

Table 14.1-1 Outline of Chapter 14: Post-Processors

Section	Processor	Function
14.2	ARGx	Interactive-graphics model/solution display; especially designed for analyses with adaptive refinement (AR)
14.3	HDBprt	High-level database printout processor
14.4	PST	COMET-AR_to_PATRAN translation (+ archival fns.)

14.2 Processor ARGx (Adaptive Refinement Graphics)

14.2.1 Overview

In this section, the graphical post-processor for adaptive mesh refinement, ARGx, and its capabilities and usage will be described.

14.2.2 ARGx — Architecture Overview

ARGx is a general purpose graphical post-processor with special provisions included for post-processing and controlling the adaptive mesh refinement environment. ARGx is fully integrated with the COMET-AR database system through high level database utilities (HDB), and may be used for either model verification purposes or as a post-processor for any model defined in the COMET-AR system contours. The general architecture of ARGx is shown in Figure 14.2-1.

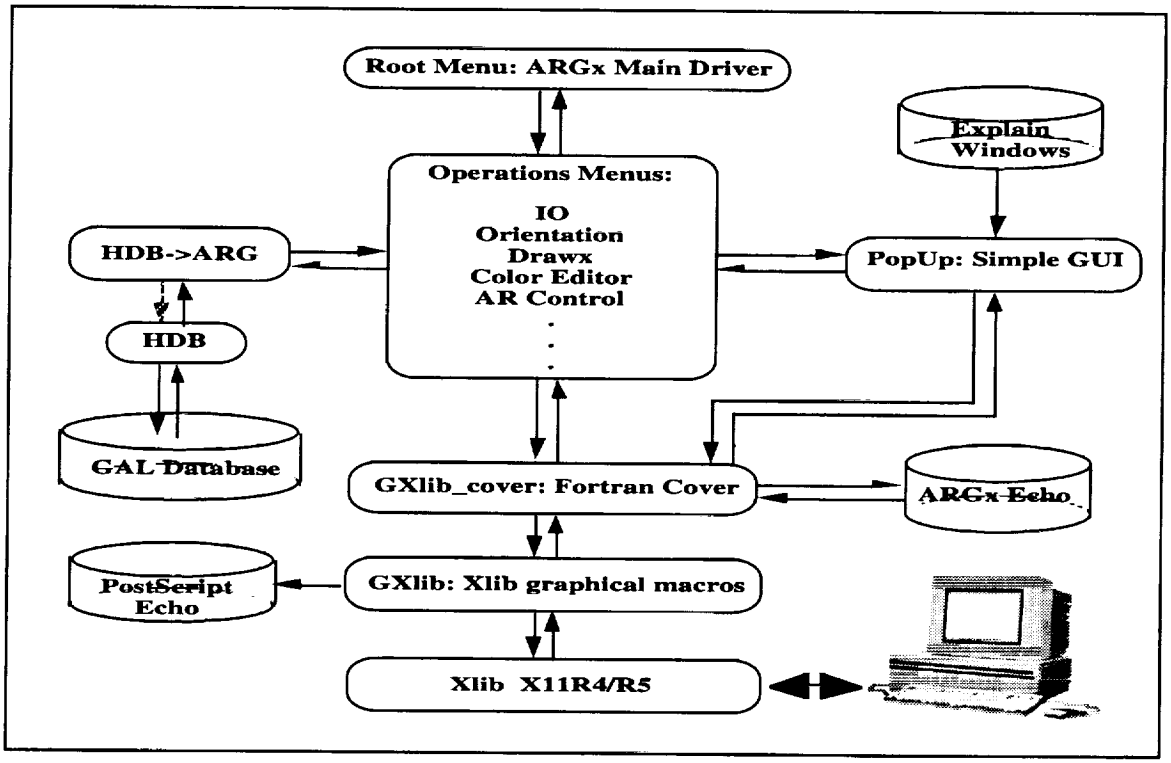


Figure 14.2-1 ARGx — Architecture

There is a direct two-way connection between ARGx and the COMET-AR database (labeled GAL database in Figure 14.2-1). ARGx contains a simple built-in graphical user interface (GUI), which interacts directly with the lowest level of the X11R4/R5 routines, the Xlib level, and it does not rely on any third-party GUI (such as Motif). This makes ARGx fully transportable to any X-platform without any modification to its source code.

The ARGx window layout is shown in Figure 14.2-2.

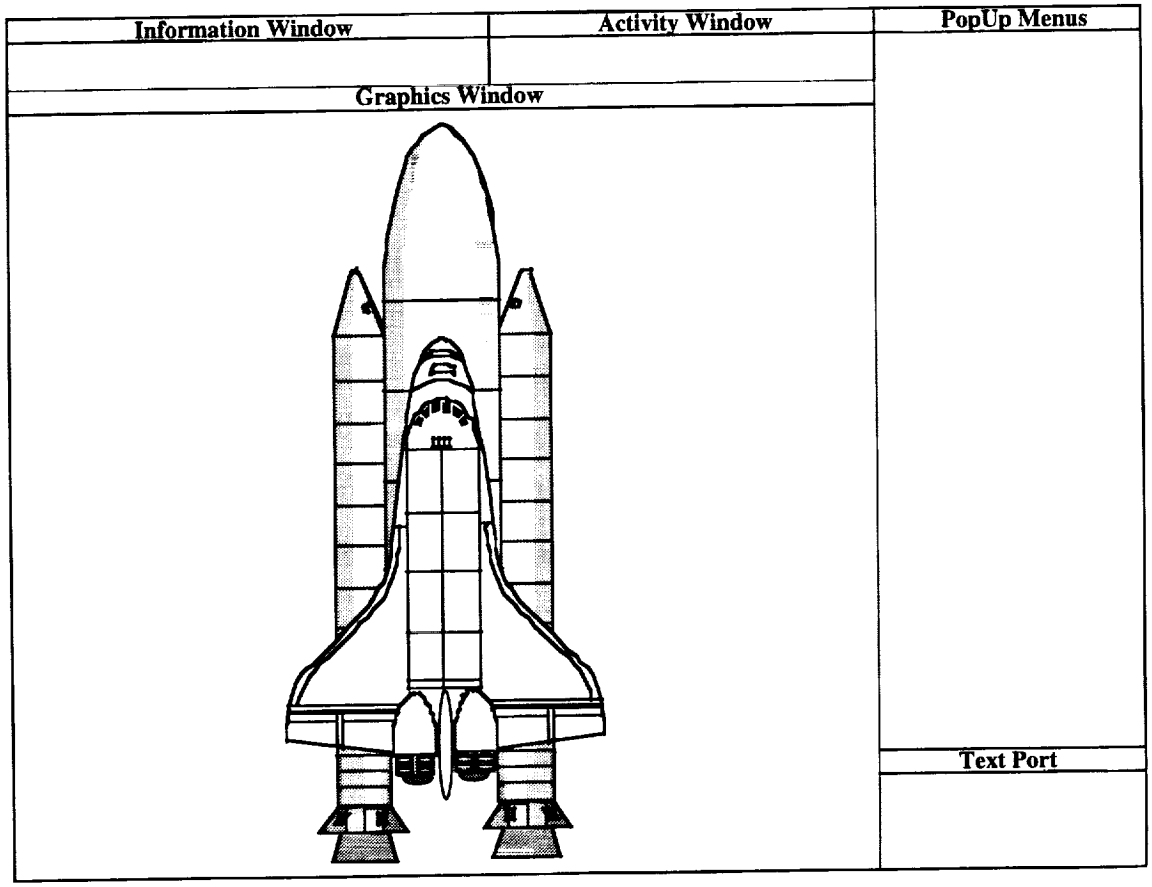


Figure 14.2-2 ARGx — Window Layout

The main portion of the display is occupied by a large square Graphics Window. In this window ARGx will display the model during the model orientation stage. This window is expanded to the entire screen when ARGx displays any type of results data. On top are, from left to right, the Information Window, the Activity Window, and the PopUp Menus windows. The Information Window is used by ARGx to display messages. Watch this window carefully when using ARGx. The Activity Window is used by ARGx for reporting on the activity being performed. This window will change its appearance to reverse video when ARGx is busy processing data and back to regular video when ARGx is waiting for some user input. The PopUp Menus window is where most of the user interface activity takes place. In this window, ARGx displays its menus and responds to user selections. At the bottom right corner, a small Text Port window is used by ARGx to graphically echo user keyboard inputs.

The menu organization within ARGx is shown in Figure 14.2-3.

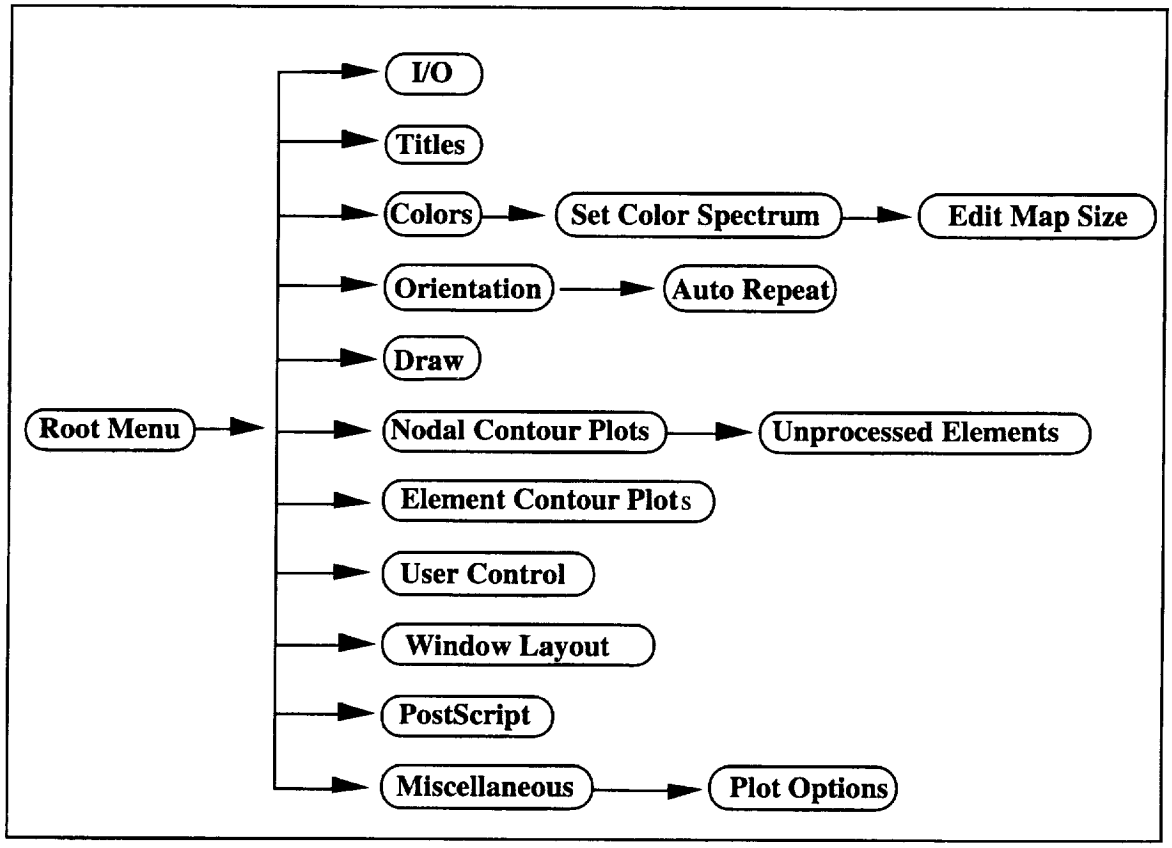


Figure 14.2-3 ARGx — Menu Organization

The Root Menu is the driver level of ARGx. It controls the flow of the program and directs user interaction. The level of menus below the Root Menu is designed to fulfill special post-processing tasks and is organized in logical order.

The unique post-processing features in ARGx are summarized below.

	Unique Capability
1	Completely integrated with COMET-AR
2	Flexible window layout (full zoom, local/global, four windows)
3	Advanced orientation tools (drag, mirror, dynamic rotations)
4	Full color spectrum control (color editor, dynamic color editor)
5	Model verification tools (BCs, loads, shell normals & triads, equation/node/element identification tools)
6	Advanced data visualization tools (fringe plots of any field, pointwise data, instant xy-plot along straight lines, and curvilinear plots along user-defined curves)
7	EPSI PostScript echo and journal file echoes
8	Fully implemented using low level Xlib—portable

14.2.3 ARGx Menu Documentation

In the following subsections the documentation of ARGx is presented in its entirety. The terms used to describe user interactions in this section are listed below.

Term	Explanation
LMB	Left mouse button
MMB	Middle mouse button
RMB	Right mouse button
CLICK	Press and immediately release a mouse button
POINT	Position the pointer over specific location without clicking a mouse button
HIGHLIGHT	Position the pointer over a menu item without clicking a mouse button
SELECT	Position the pointer over an item and click a mouse button

The following mouse button settings are the default for use in ARGx unless otherwise indicated (as in IDENTIFY and IDENTIFY/XYplot modes).

Mouse Button Setup	
LMB	Select mouse button
MMB	Ignore
RMB	Explain mouse button

14.2.3.1 ARGx — Root Menu

ARGx starts by displaying the Root Menu which is displayed in the PopUp Menus window at the top of the right side of your screen. The Root Menu enables you to control execution of ARGx. The choices available in the Root Menu are listed below.

Menu Items List	
1) I/O Menu	7) Element Contour Menu
2) Titles Menu	8) User Control Menu
3) Color Menu	9) Window Layout Menu
4) Orientation Menu	10) PostScript Menu
5) Draw Menu	11) Miscellaneous Menu
6) Nodal Contour Menu	12) Exit ARGx

Select a menu item by clicking the left mouse button (LMB). Clicking the right mouse button (RMB) over a menu item displays online help in an Explain Window for that item. The Information Window marked at the top of the left side of your screen gives you instructions during ARGx execution and the Activity window continuously updates you about current activity.

Always start by selecting the I/O Menu to open your COMET database file; set the mesh number, appropriate load set and constraint set IDs. Selecting the Orientation Menu enables you to orient the model on your display. The Draw Menu provides options to draw the deformed or undeformed configuration for the current Mesh/Step and causes all subsequent contours to be displayed with this configuration. The Titles and Color Menus enable you to edit and customize your contours, which will be generated by the Nodal or Element Contour Menu options. The PostScript Menu enables you to save PostScript Echo files (in EPSI format—Encapsulated PostScript + preview bitmap header), which you may either send to any PostScript printer or incorporate into your PostScript documents. The Exit ARGx item terminates execution of the program.

14.2.3.1.1 I/O Menu

The I/O Menu is used for selecting the GAL database file to be processed by ARGx and to set the mesh number, load step (for nonlinear analysis), load set and constraint set numbers (for linear analysis) associated with the dataset names which will be processed.

14.2.3.1.2 Titles Menu

The Titles Menu is used for setting the main title and subtitle for the various contours.

14.2.3.1.3 Color Menu

The Color Menu provides several options for setting the color pallets employed by ARGx. These options consist of a full-color map editor, color map inversion, background/foreground color inversion, and color map size (enabling control of the speed of graphical display and size of the PostScript Echo files).

14.2.3.1.4 Orientation Menu

The Orientation Menu offers a variety of tools for positioning your model in the display.

Obtaining the optimal view of your model is simple:

1) Use the rotation items in this menu for rotating the model about the principle axes (screen and world coordinate systems);

2) Once you have the proper viewing angle of your model you may want to use the Zoom item to magnify part of the model.

14.2.3.1.5 Draw Menu

The Draw Menu enables you to display the model in either its original configuration or its deformed state. This menu also provides a quick redraw capability.

14.2.3.1.6 Nodal Contour Menu

The Nodal Contour Menu enables you to select a contour based on nodal point data (either primary solution field or a smoothed secondary one) for a large selection of field components (nodal stresses, strains, displacements, etc.).

14.2.3.1.7 Element Contour Menu

The Element Contour Menu enables you to select a contour based on element data for a large selection of field components (element energy, errors, gradients, etc.).

14.2.3.1.8 User Control Menu

The User Control Menu enables you to control the automatic AR procedure by graphically selecting elements/regions in which error estimation should occur and resetting refinement indicators.

14.2.3.1.9 Window Layout Menu

The Window Layout Menu enables control of the display layout. You can toggle options between Full Window Zoom and Local/Global type Zoom and between a Single View and a Four View partitioned display. This feature is not yet implemented.

14.2.3.1.10 PostScript Menu

The PostScript Menu enables you to control the PostScript Echo of the display.

14.2.3.1.11 Miscellaneous Menu

The Miscellaneous Menu offers a variety of useful tools for visual check of a model. The boundary conditions, loads, and extremum points data may be located using options in this menu.

14.2.3.1.12 Exit ARGx

The Exit ARGx item terminates execution of the ARGx program. You will have an opportunity to save a journal file of the current session prior to exiting.

14.2.3.2 I/O Menu

The I/O Menu is used to select the GAL database file to be processed by ARGx and to set the mesh number, load step, load set, and constraint set numbers associated with the dataset names which will be processed. Once a selection is made, control returns to the ARGx Root Menu.

Menu Items List
1) Enter GAL File Name
2) Set Mesh/Step Numbers
3) Exit Menu

14.2.3.2.1 Enter GAL File Name

The Enter GAL File Name item is used to open your COMET database file. Select the Enter GAL File Name item to activate the Text Port window located at the bottom of the right side of the screen and ARGx will ask you to type in the name of your model data file. You may use any legal UNIX path as part of your data file name. ARGx will inform you if it cannot locate the file.

14.2.3.2.2 Set Mesh/Step Numbers

The Set Mesh/Step Numbers item is used for setting the mesh, load set and constraint set numbers (for linear analysis), or load step number (for nonlinear analysis) to be processed. This option invokes the HDB to ARGx data convertor, which will convert the GAL data using the HDB utilities into ARGx data structures.

14.2.3.2.3 Exit Menu

The Exit Menu item returns you to the ARGx Root Menu.

14.2.3.3 Titles Menu

The Titles Menu is used for setting the main title and subtitle for the various contours. By default, ARGx employs the GAL file name as the main title and the field name being plotted as a subtitle. The date and time stamp are appended to either the default or user-provided title. Once a selection is made, control returns to the ARGx Root Menu.

Menu Items List
1) Set Main Title
2) Set Subtitle
3) Exit Menu

14.2.3.3.1 Set Main Title

Set Main Title enables you to enter the ARGx contour's main title line. Selecting the Set Main Title activates the Text Port window and prompts for your own title. ARGx default main title is automatically set to the name of the GAL file being processed.

14.2.3.3.2 Set Subtitle

Set Subtitle enables you to enter the ARGx contour's subtitle line. Selecting the Set Subtitle item activates the Text Port window and prompts for your own subtitle. ARGx default subtitle is automatically set to the name of the result field being processed. Date and time stamps are appended to the subtitle line.

14.2.3.3.3 Exit Menu

The Exit Menu item will exit to the ARGx Root Menu.

14.2.3.4 Color Menu

The Color Menu provides several options for setting the color pallets employed by ARGx. These options consist of a full color map editor, color map inversion, background/foreground color inversion, and color map size (enabling control of the size of the postscript echo files). Once a selection is made control returns to the ARGx Root Menu.

Menu Items List
1) Set The Color Map
2) Invert Color Map
3) Invert Background Color
4) Set Number of Colors
5) Use Default Color Map
6) Save Color Map
7) Use Last Saved Color Map
8) Exit Menu

14.2.3.4.1 Set The Color Map

Set The Color Map enables you to edit the colors spectrum used for contour animation sequences. Selecting the Set The Color Map item activates the colors editor which enables you to change the boundaries of the six basic maps which construct the 256 colors used for contours. The color editor is described in detail in Section 14.2.18.

14.2.3.4.2 Invert Color Map

Invert The Color Map enables you to invert the current color maps.

14.2.3.4.3 Invert Background Color

Invert The Background Color enables you to invert the current foreground/background colors.

14.2.3.4.4 Set Number of Colors Used

The Set Number of Colors Used option allows you to set the number of colors used by ARGx to define the current color map. This number is bounded by the hardware color map size limitation. The performance of ARGx and the size of the PostScript file echo are linearly proportional to the size of the color map used.

14.2.3.4.5 Use Default Color Map

Use Default Color Map option causes ARGx to reset the color map to the default Map for use in subsequent contour plots.

14.2.3.4.6 Save Color Map

Save Color Map allows you to save the color map for use in subsequent contour plots.

14.2.3.4.7 Use Last Saved Color Map

The Use Last Saved Color Map option causes ARGx to reset the color map to the last saved map. Selecting this option enables you to dynamically modify your color map during contour sessions and record the effect of the modified color map in the PostScript Echo File.

The following procedure details the steps required to dynamically modify the color map and use the result in a PostScript file:

1) Display the desired contours;

2) Select Use Last Saved Color Map from the Color Menu;

3) Select Redraw Last Picture from the Draw Menu;

4) Dynamically edit the color map to the desired appearance;

5) Select Set PostScript Echo On from the PostScript Menu;

6) Select Redraw Last Picture from the Draw Menu;

7) Select Save PostScript File from the PostScript Menu.

14.2.3.4.8 Exit Menu

The Exit Menu item will exit to the ARGx Root Menu.

14.2.3.5 Orientation Menu

The Orientation Menu provides the orientation tools for positioning your model in the display.

Menu Items List	
1) Rotate About World X-axis	10) Set Isometric Projection
2) Rotate About World Y-axis	11) Reset Transformation
3) Rotate About World Z-axis	12) Zoom
4) Rotate About Screen X-axis	13) Un-Zoom
5) Rotate About Screen Y-axis	14) Drag Model
6) Rotate About Screen Z-axis	15) Mirror Image Model
7) Interactive Rotation Bounding Box	16) Draw the Model
8) Interactive Rotation Wire-Frame Model	17) Done Orienting
9) Set Parallel Projection	

Obtaining the optimal view of your model is simple:

1) Use the rotation items in this menu for rotating the model about the principle axes (screen or world coordinate systems);

2) Once you have the proper viewing angle of your model you may want to use the Zoom item for magnifying part of the model.

Control remains in this menu until the Done Orienting item is selected, returning you to the ARGx Root Menu.

14.2.3.5.1 Rotate bout World X_i-axis

The Rotate About World X_i-axis item enables you to rotate your model about the world coordinates X_i-axis. Selecting the Rotate About World X_i-axis item activates the Text Port window located at the bottom of the right side of your screen. ARGx prompts you to enter the rotation angle in degrees.

The world coordinate system is the physical Cartesian coordinate system in which your model is defined and is presented at the lower right corner of your screen whenever you draw your model. Selecting this item alters the transformation applied to your model but it will not redraw the model.

14.2.3.5.2 Rotate About Screen X_i-axis

The Rotate About Screen X_i-axis item enables you to rotate your model about the screen coordinates X_i-axis. Selecting the Rotate About Screen X_i-axis item activates the Text Port window located at the bottom of the right side of your screen. ARGx prompts you to enter the rotation angle in degrees.

The screen coordinate system is the display coordinate system. The screen coordinate system's origin is the lower corner of the display, X being the horizontal axis, and Y being the vertical axis in the display plane. The Z axis perpendicular to the display points outwards. Selecting this item alters the transformation applied to your model but it will not redraw the model.

14.2.3.5.3 Interactive Rotation Bounding Box

The Interactive Rotation Bounding Box item enables you to interactively rotate your model bounding box about an axis perpendicular to the direction in which the pointer is moved and an angle proportional to the distance travelled by the pointer. Any mouse button click will exit this mode and ARGx will redraw the model in the desired orientation. This option draws in "GXinverse" mode. X11/Xlib does not invert a black pixel and so this option should not be used with a black background color.

14.2.3.5.4 Interactive Rotation Wireframe Model

The Interactive Rotation Wireframe Model item enables you to interactively rotate your model wire frame presentation about an axis perpendicular to the direction in which the pointer is moved and an angle proportional to the distance travelled by the pointer. Any mouse button click will exit this mode and ARGx will redraw the model in the desired orientation. This option draws in "GXinverse" mode. X11/Xlib does not invert a black pixel and so this option should not be used with a black background color.

For this option, ARGx will require data which is contained in the Line Refinement Table (LRT as generated by the refinement processor REF1). This option will not be available if the database does not contain LRT. Since this is a computer-intensive option it is restricted to discrete transformations:

- Pointer motion of less then 5 degrees rotations is ignored.
- Any pointer motion during ARGx active mode (i.e., while the message "Drawing the Model" is displayed in the Activity Window) is discarded.

14.2.3.5.5 Set Parallel Projection

The Set Parallel Projection item enables you to set the projection employed by ARGx to a parallel projection in the (0,0,-1) direction (the viewer is located along the world Z axis looking towards the origin). Parallel projection is useful for obtaining standard 2-D projections of the model.

14.2.3.5.6 Set Isometric Projection

The Set Isometric Projection item enables you to set the type of projection employed by ARGx to an isometric projection in the (-1,-1,-1) direction (the viewer is located in the first quadrant looking towards the origin). Isometric projection is useful for obtaining 3-D views of the model and is the default orientation used by ARGx.

14.2.3.5.7 Reset Transformation

The Reset Transformation item enables you to cancel all previously defined transformations and sets the model transformation matrix as the default isometric projection matrix.

14.2.3.5.8 Zoom

Zoom enables you to magnify parts of your model. Selecting the Zoom item causes ARGx to switch to its rubber band cursor mode. ARGx will ask you to point to the lower left corner of the Zoom window, then it will switch to its rubber band cursor and ask you to point to the upper right corner of the Zoom window.

The area enclosed by the rubber band cursor is magnified to fill the area labeled "local view" and shrink the model to fit into the area marked "global view." You may define a zoom within a zoom to any magnification order in either the local or global windows.

14.2.3.5.9 Un-Zoom

Un-Zoom cancels all previously defined Zooms and redraws the model in its original magnification.

14.2.3.5.10 Drag Model

Drag Model enables you to duplicate your model by dragging it along an arbitrary axis. ARGx asks you to define the Drag Axis by entering the coordinates of two points along the axis. Once the Drag Axis is defined, ARGx will prompt for a drag rotation angle in degrees and drag distance, either of which may be set to zero. To drag the deformed model, display the deformed configuration before using the Drag tool. The Drag Tool has an auto-repeat mode which repeats the drag operation. The Drag Tool works in conjunction with any other viewing tool. Contours may be shown over the dragged model and the appropriate data will automatically be mapped into the expanded image.

14.2.3.5.11 Mirror Image Model

Mirror Image Model enables you to generate a mirror image of the model around an arbitrary plane. ARGx will ask you to define the Mirror Plane by entering the coordinates of three points within the plane. Once the Mirror Plane is defined, ARGx will generate the mirror image of the model. To generate a mirror image of the deformed model, display the deformed configuration prior to using the Mirror Tool. The Mirror tool works in conjunction with any other viewing tool. Contours may be shown over the mirrored model and the appropriate data will automatically be mapped into the mirrored image.

14.2.3.5.12 Draw Model

Draw Model causes ARGx to apply the current transformation matrix to the model original coordinates and draw the model in its new orientation. Selecting the Draw Model item causes ARGx to generate new coordinates for the view and resort its polygon list. This can be time-consuming, so hang in there, ARGx will continuously inform you about its activity in the Information and the Activity windows. The Draw Model item automatically puts ARGx into the IDENTIFY mode. In the IDENTIFY mode, ARGx expects you to use the mouse buttons as follows:

Mouse Button Setup	
LMB	Exit IDENTIFY mode
MMB	IDENTIFY current cursor position

If you enter a MMB click while the cursor is pointing to a point in the model, ARGx will respond as follows:

- The polygon containing the cursor will be highlighted;
- The nearest nodal point to the cursor position will be marked;

- The nodal point and the element identity numbers will be displayed in the Information Window.

If you enter a MMB click while the cursor is pointing to a point outside the model, ARGx will respond by displaying the "CURSOR OUTSIDE THE MODEL!!!" message in the Information Window. The IDENTIFY operation may be repeated as many times as needed. The IDENTIFY mode is not activated in Four Windows or Local/Global Zoom modes.

14.2.3.5.13 Done Orienting

Done Orienting terminates the orientation session and returns control back to the Root Menu. You can always modify your model orientation by reselecting the Orientation item from the Root Menu.

14.2.3.6 Draw Menu

The Draw Menu enables you to display the model in either its original configuration or its deformed state. This menu also provides a quick redraw capability. Once a selection is made control returns to the ARGx Root Menu..

Menu Items List
1) Draw Deformed Model
2) Draw Undeformed Model
3) Redraw Last Picture
4) Exit Menu

All items in this menu automatically put ARGx into the IDENTIFY mode. In the IDENTIFY mode ARGx expects you to use the mouse buttons as follows:

Mouse Button Setup	
LMB	Exit IDENTIFY mode
MMB	IDENTIFY current cursor position

If you enter a MMB click while the cursor is pointing to a point in the model, ARGx will respond as follows:

- The polygon containing the cursor will be highlighted;
- The nearest nodal point to the cursor position will be marked;
- The nodal point and the element identity numbers will be displayed in the Information Window.

If you enter a MMB click while the cursor is pointing to a point outside the model, ARGx will respond by displaying the "CURSOR OUTSIDE THE MODEL!!!" message in the Information

Window. The IDENTIFY operation may be repeated as many times as needed. The IDENTIFY mode is not activated in Four Windows or Local/Global Zoom modes.

14.2.3.6.1 Draw Deformed Model

Draw Deformed Model causes ARGx to apply the current transformation matrix to the deformed model coordinates and draw the model in its new orientation. Selecting the Draw Deformed Model item causes ARGx to generate new coordinates for the view and to resort its polygon list. This can be time-consuming, so hang in there. ARGx will continuously inform you about its activity in the Information and Activity windows.

The Draw Deformed Model item automatically puts ARGx into the IDENTIFY mode. In the IDENTIFY mode ARGx expects you to use the mouse buttons as follows:

Mouse Button Setup	
LMB	Exit IDENTIFY mode
MMB	IDENTIFY current cursor position

If you enter a MMB click while the cursor is pointing to a point in the model, ARGx will respond as follows:

- The polygon containing the cursor will be highlighted;
- The nearest nodal point to the cursor position will be marked;
- The nodal point and the element identity numbers will be displayed in the Information Window.

If you enter a MMB click while the cursor is pointing to a point outside the model, ARGx will respond by displaying the "CURSOR OUTSIDE THE MODEL!!!" message in the Information Window. The IDENTIFY operation may be repeated as many times as needed. The IDENTIFY mode is not activated in Four Windows or Local/Global Zoom modes.

The deformation field of the model is automatically scaled so that peak deformations will be 20% of the model dimension. Repeated selection of this item enables you to reset the scale factor. Subsequent contours will be displayed over the deformed configuration of the model.

14.2.3.6.2 Draw Undeformed Model

Draw Undeformed Model causes ARGx to apply the current transformation matrix to the original model coordinates and draw the model in its undeformed configuration. Subsequent contours will be displayed over the undeformed configuration of the model. Selecting the Draw Undeformed Model item causes ARGx to generate a new coordinates for the view and to resort its polygon list. This can be time-consuming, so hang in there. ARGx will continuously inform you about its activity in the Information and the Activity windows.

The Draw Undeformed Model item automatically puts ARGx into the IDENTIFY mode. In the IDENTIFY mode ARGx expects you to use the mouse buttons as follows:

Mouse Button Setup	
LMB	Exit IDENTIFY mode
MMB	IDENTIFY current cursor position

If you enter a MMB click while the cursor is pointing to a point in the model, ARGx will respond as follows:

- The polygon containing the cursor will be highlighted;
- The nearest nodal point to the cursor position will be marked;
- The nodal point and the element identity numbers will be displayed in the Information Window.

If you enter a MMB click while the cursor is pointing to a point outside the model, ARGx will respond by displaying the "CURSOR OUTSIDE THE MODEL!!!" message in the Information Window. The IDENTIFY operation may be repeated as many times as needed. The IDENTIFY mode is not activated in Four Windows or Local/Global Zoom modes.

14.2.3.6.3 Redraw Last Picture

Redraw Last Picture is used to re-display the last picture with the current attributes (color map choice and titles). This option is very useful in generating a postscript echo of the last view. The Redraw Last Picture item automatically puts ARGx into the IDENTIFY mode. In the IDENTIFY mode ARGx expects you to use the mouse buttons as follows:

Mouse Button Setup	
LMB	Exit IDENTIFY mode
MMB	IDENTIFY current cursor position

If you enter a MMB click while the cursor is pointing to a point in the model, ARGx will respond as follows:

- The polygon containing the cursor will be highlighted;
- The nearest nodal point to the cursor position will be marked;
- The nodal point and the element identity numbers will be displayed in the Information Window.

If you enter a MMB click while the cursor is pointing to a point outside the model, ARGx will respond by displaying the "CURSOR OUTSIDE THE MODEL!!!" message in the Information Window. The IDENTIFY operation may be repeated as many times as needed. The IDENTIFY mode is not activated in Four Windows or Local/Global Zoom modes.

14.2.3.6.4 Exit Menu

The Exit Menu item will exit to the ARGx Root Menu.

14.2.3.7 Nodal Contours Menu

The Nodal Contours Menu enables you to select a contour based on nodal point data (either primary solution field or smoothed secondary one) for a large selection of field components (stresses, strains, displacements, etc.).

Menu Items List	
1) Draw Stress Contour Lines	7) Draw Max Mxy
2) Draw Max Membrane Stress	8) Draw Strain Contour Lines
3) Draw Min Membrane Stress	9) Draw Strain Energy Contour Lines
4) Draw Max Shear Stress	10) Draw Displacement Contour Lines
5) Draw Max Bending Moment	11) Exit Menu
6) Draw Min Bending Moment	

For most of the above menu items ARGx will process the Element Stress Table (EST) and operate on stresses at each element integration point, and use the element interpolation table (EIT) to obtain smoothed nodal values. ARGx will ask you to choose INTEGRATION point or BARLOW point stress data for use in its smoothing algorithm. ARGx will also ask you to identify which element groups should be processed for nodal contouring by entering: N, GID_1, GID_2,..., GID_N, where N is the number of groups and GID_1-GID_N are the group numbers to be plotted.

Once the nodal field data is obtained, ARGx displays the extreme values of that field and asks if you wish to overwrite these values (to reset the contours to a desired range) and proceed by displaying the contours for this data.

All items in this menu automatically put ARGx into the IDENTIFY/XYplot mode. In this mode ARGx expects you to use the mouse buttons as follows:

Mouse Button Setup	
LMB	Exit IDENTIFY/XYplot mode
MMB	IDENTIFY current cursor position
RMB	SET XYplot location

If you enter a MMB click while the cursor is pointing to a point in the model, ARGx will respond as follows:

- The polygon containing the cursor will be highlighted;
- The nearest nodal point to the cursor position will be marked;

- The nodal point and the element identity numbers will be displayed in the Information Window.

If you enter a MMB click while the cursor is pointing to a point outside the model, ARGx will respond by displaying the "CURSOR OUTSIDE THE MODEL!!!" message in the Information Window. If you enter a RMB click, ARGx will activate the Rubber-line Cursor. The Rubber-line Cursor will be anchored at the cursor position of the first RMB click and will follow the cursor motion (the line will be displayed in inverse video color). If you enter a second RMB click, ARGx will respond as follows:

- The Rubber-line Cursor will be replaced by a solid arrow head line from the position of the first RMB click to the position of the second RMB click;
- If the line intersects any part of the model, an X-Y plot showing the field value of all points along the line will be displayed at the upper right corner of the display.

The IDENTIFY and X-Y plots operations may be repeated as many times as needed (using MMB and RMB clicks, respectively). X-Y plots are being echoed to the postscript file if the PostScript Echo is ON. The IDENTIFY/XYplot mode can not be activated in Four Windows mode or in Local/Global Zoom mode. Positioning the pointer over the Color Index (middle right side of the display) will re-color a single contour line with the background color. Using a RMB click while the cursor is positioned over the Color Index allows you to dynamically change the color spectrum used. A second RMB click while the cursor is positioned over the Color Index will freeze the color spectrum to the current state. These Dynamic Color manipulations will not affect the postscript echo file.

14.2.3.7.1 Draw Stress Contour Lines

Draw Stress Contour Lines is used for plotting contour lines for any stress component present in the element stress table (EST).

14.2.3.7.2 Draw Max Membrane Stress

Draw Max Membrane Stress is used for plotting contour lines for the maximum membrane stress using the element stress table (EST).

$$N_{max} = \frac{N_x + N_y}{2} + \sqrt{\left(\frac{N_x + N_y}{2}\right)^2 + N_{xy}^2}$$

14.2.3.7.3 Draw Min Membrane Stress

Draw Min Membrane Stress is used for plotting contour lines for the minimum membrane stress using the element stress table (EST).

$$N_{min} = \frac{N_x + N_y}{2} + \sqrt{\left(\frac{N_x - N_y}{2}\right)^2 + N_{xy}^2}$$

14.2.3.7.4 Draw Max Shear Stress

Draw Max Shear Stress is used for plotting contour lines for the maximum membrane stress using the element stress table (EST).

$$N_{xy}^{max} = \frac{|N_x - N_y|}{2}$$

14.2.3.7.5 Draw Max Bending Moment

Draw Max Bending Moment is used for plotting contour lines for the maximum membrane stress using the element stress table (EST).

$$M_{max} = \frac{M_x + M_y}{2} + \sqrt{\left(\frac{M_x + M_y}{2}\right)^2 + M_{xy}^2}$$

14.2.3.7.6 Draw Min Bending Moment

Draw Min Bending Moment is used for plotting contour lines for the minimum membrane stress using the element stress table (EST).

$$M_{min} = \frac{M_x + M_y}{2} + \sqrt{\left(\frac{M_x - M_y}{2}\right)^2 + M_{xy}^2}$$

14.2.3.7.7 Draw Max Mxy

Draw Max Mxy is used for plotting contour lines for the maximum membrane stress using the element stress table (EST).

$$M_{xy}^{max} = \frac{|M_x - M_y|}{2}$$

14.2.3.7.8 Draw Strain Contour Lines

Draw Strain Contour Lines is used for plotting contour lines for any strain component present in the element strain table (EST).

14.2.3.7.9 Draw Strain Energy Contour Lines

Draw Strain Energy Contour Lines is used for plotting contour lines for the strain energy densities using the element strain energy table (EST).

14.2.3.7.10 Draw Displacement Contour Lines

Draw Displacement Contour Lines is used for plotting contour lines for any displacement field component.

14.2.3.7.11 Exit Menu

The Exit Menu item will exit to the ARGx Root Menu.

14.2.3.8 Element Contour Menu

Element Contour enables you to select a contour based on elements data for a large selection of field components (Energy, Errors, Gradients, etc.). Once a selection is made control returns to the ARGx Root Menu.

.

Menu Items List
1) Draw Element Energy
2) Draw Element Energy Gradients
3) Draw Element Absolute Errors
4) Draw Element Error (MAX) Ratios
5) Draw Element Error (AVE) Ratios
6) Exit Menu

Selecting any item in this menu causes ARGx to process the Element Error Table (EET) for the quantity of interest and display the model using color coded elements. Next, ARGx will ask you to identify which element groups should be processed for element contouring by entering: N, GID_1, GID_2,..., GID_N, where N is the number of groups and GID_1-GID_N are the group numbers to be plotted.

Once the element field data is processed, ARGx displays the extreme values of that field and asks if you wish to overwrite these values (to reset the contours to a desired range) and proceed by displaying the contours for this data.

All items in this menu automatically put ARGx into the IDENTIFY/XYplot mode. In this mode ARGx expects you to use the mouse buttons as follows:

Mouse Button Setup	
LMB	Exit IDENTIFY/XYplot mode
MMB	IDENTIFY current cursor position
RMB	SET XYplot location

If you enter a MMB click while the cursor is pointing to a point in the model, ARGx will respond as follows:

- The polygon containing the cursor will be highlighted;
- The nearest nodal point to the cursor position will be marked;
- The nodal point and the element identity numbers will be displayed in the Information Window.

If you enter a MMB click while the cursor is pointing to a point outside the model, ARGx will respond by displaying the "CURSOR OUTSIDE THE MODEL!!!" message in the Information Window. If you enter a RMB click, ARGx will activate the Rubber-line Cursor. The Rubber-line Cursor will be anchored at the cursor position of the first RMB click and will follow the cursor motion (the line will be displayed in inverse video color). If you enter a second RMB click, ARGx will respond as follows:

- The Rubber-line Cursor will be replaced by a solid arrow head line from the position of the first RMB click to the position of the second RMB click;
- If the line intersects any part of the model, an X-Y plot showing the field value of all points along the line will be displayed at the upper right corner of the display.

The IDENTIFY and X-Y plots operations may be repeated as many times as needed (using MMB and RMB clicks, respectively). X-Y plots are being echoed to the postscript file if the PostScript Echo is ON. The IDENTIFY/XYplot mode can not be activated in Four Windows mode or in Local/Global Zoom mode. Positioning the pointer over the Color Index (middle right side of the display) will re-color a single contour line with the background color. Using a RMB click while the cursor is positioned over the Color Index allows you to dynamically change the color spectrum used. A second RMB click while the cursor is positioned over the Color Index will freeze the color spectrum to the current state. These Dynamic Color manipulations will not affect the PostScript echo file.

14.2.3.8.1 Draw Element Energy

Draw Element Energy is used for plotting color codes for the elements energy using the Element Error Table (EET)

14.2.3.8.2 Draw Element Energy Gradients

Draw Element Energy Gradients is used for plotting color codes for the element energy gradients using the Element Error Table (EET).

14.2.3.8.3 Draw Element Absolute Errors

Draw Element Absolute Errors is used for plotting color codes for the element absolute errors using the Element Error Table (EET).

14.2.3.8.4 Draw Element Error (MAX) Ratios

Draw Element Error (MAX) Ratios is used for plotting color codes for the element error max ratios using the Element Error Table (EET). The (MAX) ratios are obtained by dividing the element average error density by the maximum average element error density and the element area.

14.2.3.8.5 Draw Element Error (AVE) Ratios

Draw Element Error (AVE) Ratios is used for plotting color codes for the element relative error ratios using the Element Error Table (EET). The element relative error ratios is obtained by normalizing the element absolute error with the root mean square of the element strain energy.

14.2.3.8.6 Exit Menu

The Exit Menu item will exit to the ARGx Root Menu.

14.2.3.9 User Control Menu

The User Control Menu enables you to control the automatic AR procedure by graphically selecting elements/region in which error estimation should occur, and to reset refinement indicators. Once a selection is made control returns to the ARGx Root Menu.

Menu Items List
1) Draw Refined Elements
2) Error Control Menu (not implemented)
3) Refinement Control Menu (not implemented)
4) Exit Menu

14.2.3.9.1 Draw Refined Elements

Draw Refined Elements is used for highlighting element edges along which refinement indicators are set using the Line Refinement Table (LRT). Selecting the Draw Refined Elements item causes ARGx to process the LRT table for the line refinement indicators and display the model with these edges highlighted in red. The LRT hashing table file, "hash.dat", must be present in the current directory for this option to work.

14.2.3.9.2 Error Control Menu

Error Control is used to control the regions in which the AMR error estimation processor (ERR1) should operate. This menu provides a variety of graphical options for selecting and defining such regions. This option is not yet implemented.

14.2.3.9.3 Refinement Control Menu

Refinement Control is used to control the regions in which the AMR refinement processor (REF1) should operate. This menu provides a variety of graphical options for setting the refinement indicators in regions of the model. This option is not yet implemented.

14.2.3.9.4 Exit Menu

The Exit Menu item will exit to the ARGx Root Menu.

14.2.3.10 Window Layout Menu

The Window Layout Menu enables you to control the layout of the display. You can toggle options between Full Window Zoom and Local/Global type Zoom, and between a Single View to Four View partitioned display. Once a selection is made control returns to the ARGx Root Menu.

Menu Items List
1) Set Four/Single Window(s) Mode
2) Next Window
3) Set Full/Local-Global Zoom Mode
4) Exit Menu

14.2.3.10.1 Set Four/Single Window(s) Mode

The Set Four/Single Window(s) Mode option toggles between a single window mode, the default, and four windows mode. In the four windows mode the display is divided into four windows, only one of which is active at a time, allowing you to display multiple views/meshes of your model simultaneously.

14.2.3.10.2 Next Window

Next Window will set ARGx focus on the next window when in Four Windows mode.

14.2.3.10.3 Set Full/Local-Global Zoom Mode

The Set Full/Local-Global Zoom Mode option toggles between full window zoom of the selected zoom area to a local/global format of the zoom area.

14.2.3.10.4 Exit Menu

The Exit Menu item will exit to the ARGx Root Menu.

14.2.3.11 PostScript Menu

The PostScript Menu enables you to control the PostScript Echo of the display. Once a selection is made control returns to the ARGx Root Menu.

Menu Items List
1) Set PostScript Echo ON/OFF
2) Save the PostScript File
3) Exit Menu

14.2.3.11.1 Set PostScript Echo ON/OFF

Set PostScript Echo ON/OFF toggles on/off the PostScript Echo Mode. The PostScript Echo is used for generating an encapsulated color PostScript image of the current view into an ASCII file named Figure#.ps where # is the figure sequential number. The postscript file is not saved until the Save the PostScript Echo File item is selected. Displaying multiple figures in Single Window Mode without saving the file for each figure will cause an overwrite, so that only the last figure echo will be present in the postscript echo file.

14.2.3.11.2 Save the PostScript File

Save the PostScript File will save the current postscript echo file named Figure#.ps where # is the figure sequential number. This option will also toggle the Set PostScript Echo ON/OFF to OFF.

14.2.3.11.3 Exit Menu

The Exit Menu item will exit to the ARGx Root Menu.

14.2.3.12 Miscellaneous Menu

The Miscellaneous Menu offers a variety of useful tools for a visual check of the model. The boundary conditions, loads, and extremum points data may be located using options in this menu. Once a selection is made control returns to the ARGx Root Menu.

Menu Items List
1) Set Boundary Conditions Mask
2) Show Boundary Conditions
3) Show External Loads
4) Identify Min/Max Locations
5) Initialize Curve Definition
6) Add/Remove Points From Curve
7) Plot Options Menu
8) Exit Menu

14.2.3.12.1 Set Boundary Conditions Mask

The Set Boundary Conditions Mask option enables you to set the Mask for each DOF direction such that selective display of boundary conditions may be displayed. The mask is entered as an integer vector (NDOF) containing 1 for unmasked DOF and 0 for a masked one (single record in FREE format).

14.2.3.12.2 Show Boundary Conditions

The Show Boundary Conditions option graphically presents the boundary codes of the model, using single-headed arrows for displacement DOFs and double-headed arrows for rotational DOFs. Selecting the Show Boundary Conditions item causes ARGx to process the Nodal Definition Table (NDT) and extract the appropriate boundary codes. This option does not work in Four Windows Mode or Local/Global Zoom mode

14.2.3.12.3 Show External Loads

The Show External Loads option graphically presents the loads on the model, using single-headed arrows for forces and double-headed arrows for moments. Selecting the Show External Loads item causes ARGx to process the nodal external forces dataset (NVT) and extract the appropriate loads. This option does not work in Four Windows Mode or Local/Global Zoom mode.

14.2.3.12.4 Identify Min/Max Locations

Identify Min/Max Locations displays in the information window the locations (element ID and node ID) for the minimum and maximum field values for the last contours.

14.2.3.12.5 Initialize Curve Definition

Initialize Curve Definition initializes the curve definition stack. The curve definition stack is a collection of nodal point IDs used to define a curve for 2-D plotting purposes (using the Add/ Remove Points From Curve item).

14.2.3.12.6 Add/Remove Points From Curve

Add/Remove Points From Curve puts ARGx into Curve Definition mode. In the curve definition mode you can define/edit a curve through a number of nodal points in your model.

In Curve Definition mode ARGx will interpret the mouse buttons as follows.

Mouse Button Setup	
LMB	Add a point to the curve definition stack
MMB	Exit Curve Definition mode
RMB	Remove a point from the curve definition stack

The curve definition stack is limited to 1000 points.

14.2.3.12.7 Plot Options Menu

The Plot Options Menu provides options for displaying a 2-D curve plot (see Subsection 14.2.3.13).

14.2.3.12.8 Exit Menu

The Exit Menu item will exit to the ARGx Root Menu.

14.2.3.13 Plot Options Menu

The Plot Options Menu provides five options for displaying a 2-D curve plot.

Menu Items List
1) Color Linear X-Y plot
2) Linear X-Y plot
3) Curvilinear plot
4) Curvilinear color plot
5) All four options simultaneously

- The curve definition may be edited using the Add/Remove Points From Curve from the Miscellaneous Menu.
- Different solution fields may be plotted without repeating the definition of the curve. This is done by first selecting the new field to be presented from either the Nodal Contours or the Element Contours menus followed by selection of this option.
- Any of the curve plot options can be echoed into the postscript echo file by setting the PostScript Echo Mode to ON (from the PostScript Menu) prior to plotting;

When any of the color plot options is activated, the following options are available:

- Positioning the pointer over the Color Index (middle right side of the display) will re-color a single contour line with the background color.
- Using a RMB click while the cursor is positioned over the Color Index will allow you to dynamically change the color spectrum used. A second RMB click while the cursor is positioned over the Color Index will freeze the color spectrum to the current state.

For the Curvilinear plots, the following options are available:

- The Curvilinear plot may be rotated using the standard rotation tools from the Orientation Menu and redrawing the model prior to reselection of this option.
- The orientation of the curvilinear coordinates used depends on the definition of the curve. Changing the order of the definition nodes (i.e., from clockwise to counter-clockwise) will flip the direction of the normal curvilinear axis.

14.2.3.13.1 Linear X-Y Plot

Linear X-Y Plot displays a 2-D plot for the current curve defined by the curve definition stack. In this mode, the horizontal axis represents the arclength of the curve, scaled to the [0,1] interval, and the vertical axis represents the last solution field value processed by ARGx.

14.2.3.13.2 Color Linear X-Y Plot

Color Linear X-Y Plot displays a 2-D plot for the current curve defined by the curve definition stack. In this mode, the horizontal axis represents the arclength of the curve, scaled to the [0,1] interval, and the vertical axis represent the last solution field value processed by ARGx. In this option, the area enclosed between the plot curve and the horizontal axis is color-coded, filled with colors associated with the field value of the point, and a color index is displayed.

14.2.3.13.3 Curvilinear Plot

Curvilinear Plot displays a 2-D plot for the current curve defined by the curve definition stack. In this mode, a 2-D projection of the curve (into the display) is used as the tangent curvilinear axis and the normal curvilinear axis represent the last solution field value processed by ARGx.

14.2.3.13.4 Color Curvilinear Plot

Color Curvilinear Plot displays a 2-D plot for the current curve defined by the curve definition stack. In this mode, a 2-D projection of the curve (into the display) is used as the tangent curvilinear axis and the normal curvilinear axis represent the last solution field value processed by ARGx. In this option, the area enclosed between the plot curve and the actual curve is color-coded, filled with colors associated with the field value of the point, and a color index is displayed.

14.2.3.13.5 All Four Options

All Four Options displays all four options for curve plotting simultaneously. In this mode the display will be divided into four windows each displaying one of the above options.

14.2.3.13.6 Exit Menu

The Exit Menu item will exit to the ARGx Root Menu.

14.2.3.14 Auto-Repeat

The Auto-Repeat Menu enables you to set the Repeat flag on or off by selecting either the Repeat or the Stop items. Selecting the Repeat item from this menu enables you to repeat the last operation and the Stop item will terminate the Auto-Repeat Mode.

Menu Items List
1) Repeat
2) Repeat N Times
3) Stop

14.2.3.14.1 Repeat

Repeat instructs ARGx to repeat the last operation.

14.2.3.14.2 Repeat N Times

Repeat N Times instructs ARGx to repeat the last operation N times.

14.2.3.14.3 Stop

Stop instructs ARGx to terminate the Auto-Repeat mode of operation.

14.2.3.15 Set The Color Map Menu

The Set Color Map Menu enables you to construct a color map for light source shading or to collect several such maps into a vivid contours display. ARGx contains 56 predefined maps for all the possibilities of construction of a color map ramp between Black, White, Red, Yellow, Green, Turquoise, Blue, and Violet. Up to 56 such maps may be collected together, in any order, for construction of the contour lines color spectrum. For animation sequences with the Light Source Shading option, only the first map in the list is used for constructing the shading map.

This menu is contained in two sub menus, each containing 28 possible color maps. Selected maps are marked by their position within the spectrum displayed in parentheses. Selecting a marked map will remove this map from the list. After the construction of the spectrum is finished, the individual maps in the selected spectrum can be edited for size with the Edit Map Size menu, which is automatically invoked for each color map.

Menu Items List
1) From Color a To Color b
2) Next Maps List
3) Previous Maps List
4) Done Setting Spectrum

14.2.3.15.1 From Color a To Color b

From Color a To Color b items are the available color maps for constructing the contour lines color spectrum. An item name followed by a number enclosed in parentheses indicates a selected map (the number is its position in the color spectrum). Selecting an item again causes the removal of this map from the spectrum.

14.2.3.15.2 Next Maps List

Next Maps List displays the second submenu of color maps names.

14.2.3.15.3 Previous Maps List

Previous Maps List displays the first submenu of color maps names.

14.2.3.15.4 Done Setting Spectrum

Done Setting Spectrum signals ARGx that the spectrum maps list is complete and activates the Edit Map Size menu.

14.2.3.16 Edit Map Size Menu

The Edit Map Size Menu is automatically invoked by the Edit Colors menu whenever you define a color spectrum with multiple color maps. The Edit Map Size is sequentially invoked for each map in the list and enables you to change the size of that map within the color spectrum. This menu invokes an update of a displayed color spectrum map for immediate feedback.

Menu Items List
1) Increase
2) Decrease
3) Done Editing

14.2.3.16.1 Increase

Increase enables you to increase the size of a color map. Increasing a map will proportionally decrease all other maps in the list; however, a minimum map size (2 map entries) is automatically maintained.

14.2.3.16.2 Decrease

Decrease enables you to decrease the size of the color map. Decreasing a map will proportionally increase all other maps in the list; however, a minimum map size (2 map entries) is automatically maintained.

14.2.3.16.3 Done Editing

Done Editing signals ARGx that you are finished changing the currently edited map.

14.2.3.17 Unprocessed Elements Mode Menu

The Unprocessed Elements Mode Menu enables you to select the drawing mode for unprocessed elements. This menu will automatically be activated when you choose to process element data by groups (e.g., not all elements are being processed for contours).

Menu Items List
1) Display As Solid Filled
2) Display As Transparent
3) Omit

14.2.3.17.1 Display As Solid Filled

Display As Solid Filled causes ARGx to display any element for which contour data was not processed as a solid filled element (i.e., filled with the flat background color).

14.2.3.17.2 Display As Transparent

Display As Transparent causes ARGx to display any element for which contour data was not processed as a transparent element (i.e., represented as a wireframe).

14.2.3.17.3 Omit

The Omit item causes ARGx to omit from the display any element for which contour data was not processed.

14.2.4 ARGx — Advanced Usage

This section is aimed at the more advanced ARGx user. It contains examples of how to use the more sophisticated features in ARGx.

Subsection	Topic
14.2.4.1	How to Change the Automatic Deformation Scale Factor
14.2.4.2	Using the Mirror and Drag Tools
14.2.4.3	Special Visualization Capabilities
14.2.4.4	Using the Curve Plot Capabilities
14.2.4.5	Using the Color Editor
14.2.4.6	Using the Dynamically Edited Color Map in PostScript
14.2.4.7	Tips on Using PostScript Echo
14.2.4.8	Processing by Groups - When, Why and How?

14.2.4.1 How to Change the Automatic Deformation Scale Factor

The deformed model is displayed by selecting Draw Deformed Model from the Draw menu. ARGx automatically sets the deformation scale factor using:

$$ScaleFactor \times \frac{|lim(D)|}{|lim(X)|} = 0.20$$

where

$$lim(D_i) = max(D_i) - min(D_i)$$

You can overwrite this automatically computed scale factor by selecting Draw Deformed Model from the Draw menu. This time ARGx will display the current value of the displacements scale factor and ask if you wish to change this value.

14.2.4.2 Using the Mirror and Drag Tools

The following example illustrates the proper use of the Mirror and Drag tools:

1) Select Mirror from the Orientation menu;
2) Define the mirror plan by entering 3 points in the plan: (300,0,0); (300,1,0); (300,0,1);
3) Select Drag from the Orientation menu;
4) Define the drag axis by entering two points along the line: (0,0,0); (0,1,0);
5) Enter the drag distance followed by the drag rotation angle (in degrees): 300; 0;
6) The Auto Repeat menu will pop up and the drag operation can be repeated N times (without intermediate drawings).

The Mirror and Drag operations may be applied to the deformed model; however, you must display the deformed model prior to using these operations.

14.2.4.3 Special Visualization Capabilities

When displaying contour plots of any of the solution fields, ARGx activates an array of hidden visualization tools. These special capabilities are summarized below:

User Action	ARGx Response
Move cursor inside the color spectrum rectangle.	Change the color map for the pointed color to black, display field value in the information window.
Click RMB inside the color spectrum rectangle.	Dynamically modify the color spectrum. The pointed color moves with the cursor and maps above and below this color will compress/expand as the pointer moves (second user RMB click anchors the color map).
Click MMB inside model.	Display and highlight pointed element ID, nearest nodal point ID, and field value.

User Action	ARGx Response
Click RMB inside model.	Activate the rubber-line cursor anchored at the pointed position and upon receiving second LMB click display xy-plot for data along the line.

14.2.4.4 Using the Curve Plot Capabilities

ARGx includes an option to plot solution data along a curve defined as a set of nodal points. The following is the procedure for using this option:

1) Select Define Curve from the Miscellaneous Menu;

2) Add nodal points to the curve definition by using LMB clicks;

3) Remove nodal points from the curve definition by using RMB clicks;

4) Use MMB click to finish editing the curve definition;

5) Display contour plot for any solution field component;

6) Select Plot Options from the Miscellaneous Menu;

7) Select the option for the plot (Display all four Options gives a good overview).

You may repeat steps 5-7 as many times as desired. PostScript echo is possible by selecting Set PostScript Echo on from the PostScript menu before Step 6 above.

14.2.4.5 Using the Color Editor

The ARGx color editor enables you to select and edit the color map used for displaying your data. The following is the procedure for using the color editor.

1) Select Set Color Map from the Colors Menu.

2) Select the color ramps which will compose the color map. Selecting the same ramp twice will deselect that ramp. Color ramps are quadratically weighted.

3) Select Done Selecting Ramps when you are finished selecting your color ramps.

4) You can now edit the individual ramps (one at a time) by selecting Increase/Decrease from the Ramp Editor. Each Increase selection will allocate 5 more colors in the color map (compressing the remaining ramps) and each Decrease will reduce the allocated space by 5.

5) Select Done Editing when you finish editing the color ramp and ARGx will open the next selected color map for editing.

14.2.4.6 Using the Dynamically Edited Color Map in PostScript

The color map can be changed with the dynamic color editor as described in Section 14.2.4.3. ARGx can be forced to use the dynamically modified color map in the PostScript echo file with the following procedure.

1) Display your contour plot.

2) Select Use Last Color Map from the Colors Menu.

3) Select Redraw Last Picture from the Draw Menu.

4) Dynamically edit your color map using pairs of RMB clicks.

5) Select Set PostScript Echo On from the PostScript Menu.

6) Select Redraw Last Picture from the Draw Menu.

7) Select Save PostScript File from the PostScript Menu.

14.2.4.7 Tips on Using PostScript Echo

Neither the PostScript language nor Xlib have any direct provisions for shaded polygon fill (Phong or Gauraud). As a result, shading effects need to be emulated in ARGx using the supported flat-fill capabilities, which dramatically increases the size of the PostScript echo file. The following are some useful tips for controlling the size of the PostScript echo files.

- The size of the PostScript Echo file is linearly proportional to the number of colors used in the color map (256 by default, 0&1 reserved). Reduce the number of colors used prior to starting PostScript echo (64 is a nice compromise).
- Once PostScript echo is set to ON, every single graphical operation is echoed. Get used to toggling the echo ON/OFF to eliminate echoing of unnecessary operations. The triplet: Set PostScript Echo ON; Redraw Last Picture; Set PostScript Echo OFF is useful in this regard.
- In Four Windows mode, PostScript echo may be started only from the first window (lower left quadrant). Don't forget to toggle echo when moving from one window to another.

14.2.4.8 Processing by Groups — When, Why, and How

There are two occasions when ARGx inquires which element groups you wish to process:

1) When reading the model data, you can reduce the amount of data processed by ARGx by defining a list of element group IDs. For example, in the HSCT all internal structure may be eliminated by selecting only skin groups. Internal structure may be displayed by selecting only rib and spar group IDs.

2) When ARGx processes element data for contour plots (stresses, etc.), you can process the data for a subset of your model (say, skin elements only in the HSCT example). In this case nodal smoothed values will be computed based on the element group list only.

The items in 1) and 2) above are used for different purposes by ARGx and can be unrelated lists. Fringe plots will be displayed only for those elements which belong to both lists. You have the option to Omit/WireFrame/SolidFill elements in the first list but not in the second list.

14.3 Processor HDBprt (Database Print Utility)

14.3.1 General Description

The high level database print processor, HDBprt, prints any of the HDB data objects currently defined and used by COMET-AR (see Reference [1]). The purpose of this processor is to enable the COMET-AR user to obtain meaningful, labelled printouts of data objects created via the HDB utilities, as opposed to the generic, unlabeled printouts that are obtained via the *PRINT directive provided within the COMET-AR architecture described in Reference [1]. The print commands documented here employ the class-specific print utilities (*Class***prt**).

14.3.2 Processor Command Summary

To use processor HDBprt to print one or more data objects resident in a COMET-AR database, the library containing the objects to be printed must first be opened using the CLIP *OPEN directive. HDBprt allows several libraries to be open simultaneously.

Output from HDBprt can be redirected to a file by changing the logical unit number associated with CLIP's standard print device. This is achieved by using the *SET UNIT directive as follows:

*SET UNIT PRT = *prt-lun*

This directive associates the print output device with the FORTRAN logical unit number given symbolically by *prt-lun*. When the logical unit number is omitted from the *SET UNIT directive, CLIP's default print logical unit number is reinstated. Next enter the SET OBJECT command to identify a particular dataset containing the object to be printed. (Processor HDBprt knows the class of the specified object since the object class is self-described in the database.) Finally, the PRINT command can be issued repeatedly to print all or part of the selected data object.

The remainder of the commands available to processor HDBprt are of secondary importance but may be useful in some situations. A summary of the commands that are currently implemented in processor HDBprt is given in Table 14.3-1.

The *dataset_identifier* field in Table 14.3-1 is defined as follows:

dataset_identifier :== [*ldi*] { *dataset_sequence_number* | *dataset_name* }

where the (optional) *ldi* parameter specifies the GAL library containing the HDB object(s) to be treated, and either *dataset_sequence_number* or *dataset_name* must be used to identify an active (enabled) HDB-object dataset by its sequence number or its dataset name.

Table 14.3-1 HDBprt Command Summary

Command Name	Function
HELP	Help User with print processor command usage Syntax: HELP [*command*]
PRINT	Print data object attribute values or properties Syntax: PRINT [/PROP] { attribute | * } { * | col_1 [col_2 | *] }
RUN	Stop HDBprt and run another processor Syntax: RUN *processor_name*
SET CSM	Set CSM object associated with print object Syntax: SET CSM *dataset_identifier*
SET OBJECT	Set the print object Syntax: SET OBJECT *dataset_identifier*
SHOW CSM	Display associated CSM object library and dataset Syntax: SHOW CSM
SHOW LIBS	Display names and LDIs of open libraries Syntax: SHOW LIBS
SHOW OBJECT	Display print object library and dataset Syntax: SHOW OBJECT
STOP	Stop HDBprt processor Syntax: STOP

14.3.3 Command Glossary

14.3.3.1 HELP Command

Help on an HDBprt command is available with the HELP command, the syntax of which is:

HELP [command]

14.3.3.2 PRINT Command

The PRINT command displays information for selected attributes in the data object (chosen with the SET OBJECT command) and has the following syntax:

PRINT [/PROP] { *attribute* | * } { * | col_1 [col_2 | *] }

Object information consists of two types: attribute properties and attribute values. Attribute properties consist of attribute name, data type, current dimension, maximum dimension, and index in the physical record. Attribute properties are displayed when the PROP qualifier is specified; otherwise attribute values are displayed.

The *attribute* argument in the PRINT command is the name of an attribute to be printed. All object attributes may be printed by entering an asterisk (*) in place of an attribute name. All columns are printed by using an asterisk (*) in the final field of this command. If a column range is specified, then the col_1 argument is the logical table column number of the first column to be printed and the col_2 argument is the logical table column number of the last column to be printed. If an asterisk (*) is specified in place of the col_2 value, then all columns from col_1 to the last physical column will be printed. Column argument values are not applicable when printing attribute properties.

14.3.3.3 RUN Command

Processor HDBprt can be terminated and another processor activated with the RUN command:

RUN *processor_name*

Processor HDBprt can also be terminated with the STOP command.

14.3.3.4 SET CSM Command

Most HDB data objects must be associated with a CSM data object when they are opened. Processor HDBprt uses either the default CSM object, or one that is specified using the SET CSM command:

SET CSM *dataset_identifier*

where the syntax for *dataset_indentifier* is given by:

dataset_identifier :== [*ldi*] { *dataset_sequence_number* | *dataset_name* }

If the library logical device index (*ldi*) is not specified, then the logical device index of the last SET CSM object will be used as the default value.

If a CSM object is not set using the SET CSM command, processor HDBprt assumes that the CSM data object to be associated with the print object is in the same library as the print object and has the *dataset_name* CSM.SUMMARY.0.0.*mesh*, where *mesh* is the same mesh number as that specified for the print object.

14.3.3.5 SET OBJECT Command

Before an HDB object (also called the print object) can be printed, it must be specified using the SET OBJECT command:

SET OBJECT *dataset_indentifier*

where the syntax for *dataset_indentifier* is given by:

dataset_identifier :== [*ldi*] { *dataset_sequence_number* | *dataset_name* }

If the library logical device index (*ldi*) is not specified, then the logical device index specified in the last SET OBJECT command will be used as the default value.

14.3.3.6 SHOW CSM Command

Library and dataset names for the currently set CSM data object can be displayed with the SHOW CSM command:

SHOW CSM

14.3.3.7 SHOW LIBS Command

Before an HDB data object can be printed, the library on which it resides must first be opened using the CLIP *OPEN directive. To display the names and logical device indices of all data libraries currently open (since several libraries can be open at once), use the SHOW LIBS command:

SHOW LIBS

When an open library is no longer required, it can be closed with the CLIP *CLOSE directive.

14.3.3.8 SHOW OBJECT Command

The library and dataset name for the currently set print object can be displayed with the SHOW OBJECT command:

SHOW OBJECT

14.3.3.9 STOP Command

Processor HDBprt can be terminated with the STOP command:

STOP

Processor HDBprt can also be terminated by executing another processor with the RUN command.

14.3.4 Database Input/Output Summary

Processor HDBprt makes no assumptions regarding dataset names, so long as they follow standard naming conventions. It is assumed that any dataset whose name is supplied (as part of a *dataset_identifier*) to a SET CSM or a SET OBJECT command contains an HDB data object. Since HDB data objects are self-descriptive, HDBprt not only knows whether the user supplied dataset contains an HDB object, but exactly to which class the data object belongs.

For a summary of the possible input dataset names for datasets (containing HDB data objects) that HDBprt can print, refer to Chapter 15, *Database Summary,* or consult Reference [1] for more details. The dataset names found there follow the suggested naming conventions for COMET-AR data objects, but actual datasets may be named according to the user's preference.

The HDBprt processor does not create any output datasets.

14.3.5 Limitations

Processor HDBprt has no limitation other than the ability to print only HDB data objects.

14.3.6 Error Messages

The most commonly occurring error messages printed by HDBprt are presented in Table 14.3-2. Each message has an associated probable cause and a recommended user action.

Table 14.3-2 Processor HDBprt Error Messages

Index	Error Message	Probable Cause	Recommended Action
1	Undefined Print Object Library or Dataset	*Print object* has not been set	Use SET OBJECT command
2	Undefined Default CSM Library or Dataset	CSM object has not been set and HDBprt cannot find one with the default name	Use *TOC to find the appropriate CSM object; use SET CSM command to set the data object
3	Undefined Dataset	Bad dataset name or sequence number on a SET command	Use *TOC to check for correct name or number; re-enter SET CSM or SET OBJECT command
4	Library Is Not Open	Bad library LDI on a SET command	Use SHOW LIBS to see open libraries; use *OPEN to open a new library
5	Bad LDI Value	Library LDI value is out-of-range	Acceptable LDI values are in the range 1–32
6	Dataset Identifier Cleared	SET CSM/OBJECT is entered with no *dataset_identifier*	Supply a proper *dataset_identifier*
7	Bad Dataset Identifier Syntax	Badly formed *dataset_identifier*	Use HELP SET CSM for proper *dataset_identifer* syntax
8	CSM Open Failed	Cannot open a CSM data object	Check CSM library LDI and dataset name

Table 14.3-2 Processor HDBprt Error Messages (Continued)

Index	Error Message	Probable Cause	Recommended Action
9	Bad PROP Value	Unrecognized PRINT qualifier	Check spelling of /PROP qualifier
10	Bad CSM CONTENTS Record	Reference to a non-CSM dataset	Check spelling of dataset name or correctness of dataset sequence number
11	Bad ATTRIBUTE Value	Attribute name on PRINT is not text or an '*'	Check spelling of attribute name
12	Undefined Attribute Name	Name is not an attribute of the print object	Check spelling of attribute name
13	Bad COL1 Value	Argument col_1 on PRINT is not an integer or an '*'	Check argument col_1
14	Bad COL2 Value	Argument col_2 on PRINT is not an integer or an '*'	Check argument col_2
15	Too Many Command Arguments	Bad command syntax	Use HELP *command*
16	Bad CONTENTS Record	Reference to a non-HDB data object	Check spelling of dataset name or correctness of dataset sequence number

14.3.7 Example and Usage Guidelines

The following is a typical command script for the HDBprt Processor.

```
*open 1 shuttle.dbc        . open library on LDI 1
*toc                       . look at dataset names
set obj 1 csm.summary...4  . set CSM for mesh 4
print * 1                  . print all CSM attributes
set obj 1 37               . set NDT for mesh 2
print * 4 10               . print NDT for nodes 4..10
*set unit prt = 3          . set output file to FORTRAN unit 3
print * *                  . print complete Nodal DOF Table
*set unit prt              . set CLIP's default output file
print state 1 20           . print NDT Table STATE attribute
...                        . more HDBprt & CLIP commands
stop                       . exit HDBprt
```

After a library (shuttle.dbc) is opened on CLIP's logical device index 1 and its table of contents (TOC) examined, the CSM data object for mesh 4 is SET and its first column printed. Next, the Nodal DOF Table for mesh 2 (given by the TOC as dataset sequence number 37, for example) is SET and all of its attributes in columns 4 through 10 are printed. This example assumes an interactive session with HDBprt output is going to the user's display. The next command (*SET UNIT directive) redirects future processor output to a permanent file associated with FORTRAN unit 3. The following PRINT command prints the entire NDT data object to this file. After restoring the print device to be the user's display, the STATE attribute in columns 1 through 20 of

the NDT data object is printed. Finally, after the user has completed examination of library shuttle.dbc, the HDBprt processor is terminated with the STOP command.

14.3.8 References

[1] Stanley, G. and Swenson, L., *HDB: Object-Oriented Database Utilities for COMET-AR,* NASA Computational Structural Mechanics (CSM) Contract Report, August 1992.

14.4 Processor PST

14.4.1 General Description

PST consists of a set of sub-processors which perform various pre- and post-processing functions. Many of these functions were developed independently of each other in order to fulfill the pre- and post-processing requirements that appeared while validating Adaptive Mesh Refinement (AR). Over time these functions have been collected and integrated into PST (PoST). Many commands have not been tested for general robustness, and the input and output to commands are often tailored for particular users and applications. As a result, many functions are not user-friendly. Nevertheless, PST may offer useful pre- and post-processing functions. Improvements and modifications to PST are encouraged. Table 14.4-1 summarizes all PST sub-processors.

Table 14.4-1 Summary of Sub-Processor Names

Sub-Processor Name	Function
ARCHIVE	Archives selected COMET-AR data to the database
TRANSLATE	Translates data between analyzer data formats

The ARCHIVE sub-processor extracts resultant quantities from the database at selected locations and archives/or displays those quantities in the database. The original purpose of this sub-processor was to extract and tabulate nodal adaptive mesh refinement data used to create XY-plots. Since the plot package used to create the XY plots was capable of reading fixed-length GAL records, all output from the ARCHIVE commands employ this output format.

The TRANSLATE (previously called CONVERT) sub-processor translates data between different FEM analyzers and external modeling tools. Currently the translator recognizes COMET-AR, PATRAN,[1] and STAGS 2.0[2] (QSTAGS) data formats. The translator converts data from the source data format to a temporary intermediate data format, and then to the destination data format. By adopting an intermediate data format, each additional data format added to the PST immediately has access to all other data formats.

The input to PST is governed by the COMET-AR command language CLAMP (see Reference [1]); therefore, PST accepts both CLIP directives [2] and PST-specific commands as input. The PST executive level commands are shown in Table 14.4-2. They consist of commands to execute sub-processors (usually the name of the sub-processor), SET *parameter* commands, and a STOP command.

1. The translator reads PATRAN neutral files.
2. The translator uses STAR to interrogate and create a STAGS database.

Table 14.4-2 PST Command Summary

Section	Command Name	Function
14.4.2	ARCHIVE	Archives selected COMET-AR data to the database
14.4.3	SET *parameter*	Set default names and parameters
14.4.4	STOP	Exit the PST processor
14.4.5	TRANSLATE	Translates data between analyzer data formats

A sub-processor is invoked by typing the name of the sub-processor at the main command level. Once a sub-processor has been executed, each sub-processor responds to a separate list of commands, in addition to the globally applicable SET commands. The SET commands augment the sub-processor specific commands and are used to change the value of input parameters from their default values. In general, a SET command can be issued from anywhere within PST in order to change the value of a parameter. Once a new parameter value has been set, the value applies throughout PST until the parameter is changed with another SET command. Most SET parameters have been assigned a default value.

14.4.2 ARCHIVE Sub-Processor

14.4.2.1 General Description

The ARCHIVE sub-processor extracts resultant quantities from the database at selected locations and archives those quantities in the database. The ARCHIVE sub-processor is invoked by issuing the ARCHIVE command at the PST main prompt.

```
ARCHIVE
```

This command transfers you to the ARCHIVE command level, where you can chose from a variety of archival commands as described in Table 14.4-3. Each archival command extracts a specific solution quantity or model summary parameter from the central data library (.DBC). The CPU command differs from other ARCHIVE commands, in that it operates on the output generated by a COMET-AR analysis. This output is required to be saved in a log file. The result of each archival function is stored as GAL record group in the database. By default, the results are stored on the central data library (.DBC); however, the output is typically redirected to the results data library (.DBR). The user can control the name of the output data library and dataset.

Table 14.4-3 Summary of ARCHIVE Commands

Command Name	Function
ATTRIBUTE	Retrieves and archives attributes from the CSM data object.
CPU	Reads and tabulates the CPU information recorded in a COMET-AR log file by a linear static solution with adaptive mesh refinement.

Table 14.4-3 Summary of ARCHIVE Commands (Continued)

Command Name	Function
DISPLACEMENT	Retrieves and records selected data from the NODAL.DISPLACEMENT vector. This command has the same functionality as the VECTOR command except that the displacement vector is chosen automatically.
NODES	Allows you to define a list of nodes for which data will be extracted and recorded. Alternatively, you specify a coordinate range from which PST extracts a list of nodes.
RESULTANT	Retrieves and records selected data from STRAIN, STRESS, and STRAIN_ENERGY datasets (EST data objects).
RETURN	Return to main command level.
VECTOR	Retrieves and records selected data from a user specified NVT or NAT vector.

When using the CPU, DISPLACEMENT, NODES, RESULTANT, or VECTOR command, control is first transferred to a sub-command level, which accepts additional commands specific to the ARCHIVE command in question. These sub-commands augment SET commands, and provide additional control to the archival of various solution quantities. The format of sub-commands is similar to SET commands, but they are not preceded by the keyword SET.

The functional distinction between SET commands and sub-commands is not always clear. In general, sub-commands are specific to the particular archive function chosen. The SET commands control global quantities and may apply to other ARCHIVE commands, sub-commands, or other sub-processors in PST. The actual archival of a solution quantity is initiated by typing the command ARCHIVE at the sub-command level. (ARCHIVE command appears in two situations: in one case it is the name of a sub-processor and in another case it is used to initiate the actual archival of different solution quantities.) The ATTR and RETURN command do not have additional command levels. These commands complete their functions as soon as a valid input command has been entered.

In the command descriptions that follow, all SET commands applicable to a particular ARCHIVE command are listed. For a more detailed description of each SET command refer to Section 14.4.3.

14.4.2.2 ATTRIBUTE Command

14.4.2.2.1 General Description

This command retrieves the value of a general summary attribute from the CSM data object and deposits it in a GAL record. Any general summary attribute in the CSM object with a dimension of 1 can be archived with the ATTRIBUTE command. A CSM object exists in the central data library for each mesh generated during an analysis. The CSM data object associated with each mesh is selected with the SET MESH command. The output from this command consists of GAL record groups, macrosymbols, and output to standard output. The macrosymbols are created for use within CLAMP-procedures. This command does not have a sub-command level.

This command has primarily been used to archive the *neq* attribute of the CSM data object in order to plot the global convergence of finite element solutions quantities obtained via adaptive mesh refinement.

14.4.2.2.2 Command Syntax

The ATTRIBUTE command accepts only one parameter as input, the name of a general summary attribute from a CSM data object. Once the command has been issued, the value of the attribute is retrieved from the default data library (specified via the SET LDI command) and archived in a GAL record group. The command syntax for the ATTRIBUTE command is composed of the ATTRIBUTE keyword followed by a valid attribute name:

```
ATTRIBUTE attribute
```

Valid attributes are listed in Table 2.4-3 in the HDB manual [3].

14.4.2.2.3 Relevant SET Commands

The SET commands that apply to the ATTRIBUTE command are listed in Table 14.4-4. None of these SET commands are mandatory, but the SET MESH command is usually included to indicate the meshes from which to extract attributes. For a more detailed description of SET commands, see Section 14.4.3.

Table 14.4-4 Relevant SET Commands for the ATTRIBUTE Command

Command Name	Default	Function
SET DATASET_NAME	1,ARCHIVE	Name of dataset which receives the archived quantities as a record group
SET LDI	1	Logical device index of computational data library (.DBC)
SET LDI/ARCHIVE	1	Logical device index of results data library.
SET MESH	0,0,1	Range of meshes from which data is to be archived.

14.4.2.2.4 Input Datasets

The input to the ATTRIBUTE commands is always a CSM data object. You specify which CSM date object to consider by supplying a mesh number or mesh range, which then is used to automatically search the default data library for the CSM data objects associated with the mesh numbers specified. Once the CSM data object has been found, the appropriate attribute is extracted and archived (see Table 14.4-5).

Table 14.4-5 ATTRIBUTE Command Input Datasets

Dataset	Type	Description
CSM.SUMMARY...*mesh*	CSM	Complete Summary of the Model (CSM data object)

14.4.2.2.5 Output Datasets

The output from this command consists of GAL record(s) containing the value of the attribute requested. The record has the same name as the attribute. If a mesh range is specified, a record group is formed. Each record in the record group corresponds to one mesh. The `SET MESH` command determines which record in the record group is assigned an attribute value. If a record already exists the data is overwritten with a new value. Mesh=0 corresponds to record number 1 in a record group, since DB does not allow record numbers to start from zero. In general, mesh *n* corresponds to record *n+1*. Table 14.4-6 shows the name of the ATTRIBUTE command output datasets, while Table 14.4-7 shows the name of the output records.

Table 14.4-6 ATTRIBUTE Command Output Datasets

Dataset Name	Default	Description
dataset_name	1,ARCHIVE	Name of archival dataset name

Table 14.4-7 ATTRIBUTE Command Output Records

Record Name	Description
attribute.<mesh+1>	Record containing the value of the attribute selected for a given mesh.

14.4.2.2.6 Output Macrosymbol

In addition to creating record groups, a global macrosymbol is created as in Table 14.4-8.

Table 14.4-8 ATTRIBUTE Command Output Macrosymbols

Macrosymbol Name	Description
attribute[mesh]	Macrosymbol containing the value of the attribute selected for a given mesh. (*Attribute* [0] corresponds to *mesh* 0.)

14.4.2.2.7 Limitations

This command is limited to the archival of general summary attributes from the CSM data object only. Valid attributes are listed in Table 2.4-3 in the HDB manual [1]. Element type summary attributes can currently not be archived with this command.

14.4.2.2.8 Error Messages

The error messages in Table 14.4-9 are possible with the ATTRIBUTE command.

Table 14.4-9 Summary of Errors Displayed by the ATTRIBUTE Command.

Error #	Error Message	Probable Cause(s)	Recommended User Response
1	Cannot open CSM data object ...	No CSM data object for mesh number or logical device index specified.	Check data library and set correct mesh number and logical device index.
2	Cannot access attribute from CSM data object ...	Misspelled name of attribute requested; attribute not a valid general summary attribute (with unit dimension)	Check name of attribute specified. Valid attributes are listed in Table 2.4-3 in the HDB Manual [1].
3	Problems opening the ATTRIBUTE record group.	Resultant data library not open or name of results dataset is invalid.	Open resultant data library with *OPEN directive. Specify a valid dataset name.
4	Unable to write ATTRIBUTE record.	Bad record group identifier, internal tables clobbered.	Read complete error diagnostic. If meaning is unclear see developer.
5	Unable to close CSM object	Bad HDB data object handle of data library conflict.	Read complete error diagnostic. If meaning is unclear see developer.
6	Problem closing attribute record group	Bad DB record group identifier, or dataset clobbered.	Read complete error diagnostic. If meaning is unclear see developer.

14.4.2.2.9 Examples and Usage Guidelines

The command stream shown below is used to extract two attributes from the CSM data object, *neq* and *nnode*. First, the ARCHIVE command is issued at the main PST command level to enter the ARCHIVE sub-processor. The SET commands are used to specify for which meshes to archive the attributes NEQ and NNODE, to set the name of the archival dataset, and to specify the logical device index associated with the central data library. Finally, the ATTRIBUTE command is issued, which extracts the value of the attributes from the CSM data object of the KP.DBC data library, and archives these values in record groups called NEQ and NNODE.

```
*open 1 KP.DBC
*open 2 KP.DBR

. SET commands
. ============
SET LDI  = 1
SET MESH = 0,4,1
SET DATA = 2, ARCHIVE.SUMMARY
```

```
   ARCHIVE

      ATTRIBUTE NNODE
      ATTRIBUTE NEQ

      RETURN

.  Display macrosymbols created
.  ============================
   *show macros

.  Display content of results library
.  ==================================
   *rat 2

   STOP
```

The output generated by the command stream above is shown here. In addition to creating two record groups, attribute values are defined via macrosymbols and displayed to standard output.

```
 ** BEGIN PST       ** Using Dynamic Memory **
 <DM> OPEN,  Ldi:  1, File: KP.DBC , Attr: old, Block I/O
 <DM> OPEN,  Ldi:  2, File: KP.DBR , Attr: old, Block I/O

     Mesh            NNODE
        0                 24
        1                 80
        2                244
        3                860
        4                920

     Mesh            NEQ
        0                 86
        1                338
        2               1062
        3               3950
        4               4170

<CL> lv:0 t:I CSM_PRECISION 2
<CL> lv:0 t:I NNODE[0] 24
<CL> lv:0 t:I NNODE[1] 80
<CL> lv:0 t:I NNODE[2] 244
 <CL> lv:0 t:I NNODE[3] 860
 <CL> lv:0 t:I NNODE[4] 920
 <CL> lv:0 t:I NEQ[0] 86
 <CL> lv:0 t:I NEQ[1] 338
 <CL> lv:0 t:I NEQ[2] 1062
 <CL> lv:0 t:I NEQ[3] 3950
 <CL> lv:0 t:I NEQ[4] 4170

. Content of
 Record Table of dataset ARCHIVE.SUMMARY
 Key           L_cyc  H_cyc  Type Log_size
 NEQ               1      5   I          1
 NNODE             1      5   I          1

      ENDRUN called by CLIP
 <DM> CLOSE, Ldi:  1, File: KP.DBC
 <DM> CLOSE, Ldi:  2, File: KP.DBR
```

14.4.2.3 CPU Command

14.4.2.3.1 General Description

This command was specifically created to archive the CPU time associated with a linear static solution with adaptive mesh refinement. The command reads the content of a COMET-AR log file, and scans it for the CPU information printed by every processor during an analysis. Every time the processor EXIT statement is encountered the name of the processor and the CPU time spent in that processor is recorded. From this data, the total accumulated solution time for the whole analysis, and the incremental CPU time associated with each mesh is printed to standard output and archived in the data library.

This command works by recognizing certain features in the log file. The CPU recording is initiated when the character string highlighted below is encountered. From the highlighted character string the program reads the starting mesh number and then proceeds to read and tabulate the CPU information from each processor until the last processor for that particular mesh has been encountered. The computations for a mesh (a mesh loop) consist of adaptive mesh refinement, solution, and error estimation. Since error estimation is the last step, the end of a mesh is recognized by looking for the execution of an error processor. Once the CPU information from an error processor has been encountered, the CPU for all processors of that mesh are added and summarized. Since the number of error processors called at the end of a mesh loop varies depending on the nature of the problem, the user has to specify the number of error processors executed at the end of each mesh loop. Once this information has been supplied, the CPU command is capable of summarizing the CPU history for the analysis.

```
Control Loop Summary
========================
MESH       = 0
FIRST_STEP = 0
LAST_STEP  = 0

Adaptive Mesh Refinement = 0
Solution                 = 1  L_STATIC_1
Error Estimation         = 1
Stress                   = 1

Research Parameter Summary
==========================
Mesh initialization = 0
Interpolation       = 0

########################################
<DM> OPEN,  Ldi:  2, File: KP.DBE , Attr: new, Block I/O
<DM> OPEN,  Ldi:  3, File: KP.DBS , Attr: new, Block I/O
<DM> CLOSE, Ldi:  3, File: KP.DBS, Opt: DELETE
<DM> OPEN,  Ldi:  3, File: KP.DBS , Attr: new, Block I/O
####################################

L_STATIC  Mesh 0  Step 0
```

```
####################################
======================
LINEAR STATIC ANALYSIS
======================
** BEGIN ES1       ** Using Dynamic Memory **
<DM> OPEN,  Ldi:  6, File: ES1_EX96.HIST.CONV , Attr: new, Block I/O
<DM> CLOSE, Ldi:  6, File: ES1_EX96.HIST.CONV
   Element configuration initialized.
** BEGIN ES1       ** Using Dynamic Memory **
EXIT ES1       CPUTIME=       1.19  I/O(DIR,BUF)=       0       0

UNIX COMET AR -- VERSION 2.0 -- 07_JUN_1993        (aml_32)
```

14.4.2.3.2 Command Syntax

The command syntax for the CPU command is shown below.

```
ARCHIVE
   SET FILE = log_file
   CPU
      [CALL_ERR  = call_err ]
      [CUMULATIVE = [ <true> | <false> ]]
      [RETURN]
       ARCHIVE
   RETURN
STOP
```

The CPU function is invoked by first executing the ARCHIVE sub-processor and then invoking the CPU command from the ARCHIVE sub-processor. The SET commands are used to augment the CPU commands shown in Table 14.4-10. The SET FILE command is always required in order to supply the name of the log file. The actual extraction of CPU information is initiated by entering the ARCHIVE command. In order for this command to work properly it is necessary to supply the correct value to the CALL_ERR command. This command is used to identify the end of a mesh loop. All mesh loops are assumed to end with a call to an error processor.

Table 14.4-10 Summary of CPU Commands

Command Name	Default	Function
ARCHIVE		ARCHIVE CPU times from log file
CALL_ERR	1	Number of times error processor(s) are called per mesh.
CUMULATIVE	<true>	CPU-time accumulation time. If this flag is on the CPU time of each mesh is added to the total CPU time.
RETURN		Exit from CPU command module

14.4.2.3.3 The ARCHIVE Command

This command initiates the archival process.

```
ARCHIVE
```

14.4.2.3.4 The CALL_ERR Command

This command is used to specify how many times error processors are called during a mesh loop. Determine the number of calls to error processors by counting the number of times the string "`** BEGIN ERR? ** Using Dynamic Memory **`" appears in each mesh loop of the log file, where `?` is a generic symbol for any character. For an analysis with no partitions, this number is usually one (the default). If the structure is partitioned, the call to the main error processor is automatically followed by a call to an error processor called ERRa, which accumulates the element errors in different groups into a global error. In this case *call_err* is equal to 2.

```
CALL_ERR = call_err
```

14.4.2.3.5 The CUMULATIVE Command

This command adds the CPU time associated with each mesh to the previous mesh, resulting in the display of a total accumulated CPU for each mesh. To obtain the CPU associated with each mesh inactivate this parameter by setting it to <false>.

```
CUMULATIVE = [ <true> | <false> ]
```

The default value of `CUMULATIVE = <true>`.

14.4.2.3.6 The RETURN Command

This command exits from the CPU command level and returns to the ARCHIVE command level.

```
RETURN
```

14.4.2.3.7 Relevant SET Commands

Table 14.4-11 lists all SET commands relevant to the CPU command.

Table 14.4-11 SET Commands Relevant to CPU command

Command Name	Default	Function
`SET DATASET_NAME`	`1,ARCHIVE`	Name of dataset which receives archived quantities as a record group.
`SET FILE_NAME`		Name of log file. This SET command must be supplied by the user in conjunction with the CPU command.

Table 14.4-11 SET Commands Relevant to CPU command (Continued)

Command Name	Default	Function
SET HEADER	L_STATIC Mesh	Sets the format of the header string searched to initiate CPU timing for each mesh. This SET command is only used for this command.
SET LDI	1	Logical device index of central data library (.DBC).
SET LDI/ARCHIVE	1	Logical device index for results data library (.DBR).
SET MESH	0,0,1	Range of meshes from which data is to be archived.

14.4.2.3.8 Input Datasets

There are no input datasets associated with the CPU command. Instead a file containing output from COMET-AR processors is expected.

14.4.2.3.9 Output Datasets

The output from this command consists of four GAL record groups summarizing the CPU time spent by various processors during an analysis. The names of these record groups are shown in Table 14.4-12 and cannot be changed. The format of the dataset name is shown in Table 14.4-13. The SET MESH command determines which records in the record group are assigned a CPU time. Mesh=*n* corresponds to record number *n+1*. DB does not allow record numbers to start from zero.

Table 14.4-12 CPU Command Output Datasets

Dataset Name	Default	Description
dataset_name	1, ARCHIVE	Total CPU time for of all processors of a given mesh

Table 14.4-13 CPU Command Output Records

Record Name	Description
CPUTOT.<*mesh+1*>	Total CPU used by all processors for a given mesh
CPUASM.<*mesh+1*>	Total CPU spent in assembly and RHS operations for a given mesh
CPUSKY.<*mesh+1*>	CPU for FACTOR, SOLVE and matrix decompositions steps
CPUES.<*mesh+1*>	Total CPU spent in the element processors.

14.4.2.3.10 Limitations

This command is programmed to recognize a certain format in the log file. If the format of the log file changes the corresponding change has to be made to the program source code.

This command only works in conjunction with linear static solutions.

14.4.2.3.11 Error Messages

The CPU command produces user-friendly error messages that are self-explanatory.

14.4.2.3.12 Examples and Usage Guidelines

In this example the CPU times recorded in the file ka.log are accumulated and summarized in the dataset ARCHIVE.SUMMARY which resides on the results data library KP.DBR. The input deck is shown below.

```
   *echo,off
   *open KP.DBR

.  Set General Parameters
.  ======================
   SET ECHO         = <true>
   SET FILE         = ka.log
   SET DATA         = 1,ARCHIVE.SUMMARY
   SET MESH         = 1,2

   ARCHIVE

.     Define Node to extract displacements from
.     =========================================
      CPU
         CALL_ERR     = 1
         ARCHIVE

      RETURN

   STOP
```

The output shows the CPU information for mesh 1 and 2. To obtain the CPU information, the program searches the log file ka.log for the mesh specified via the SET MESH command. If other meshes are encountered during the search, this is acknowledged and no CPU information is printed for these meshes.

```
<CL> PUT_message,Commnt>
 ** BEGIN PST       ** Using Dynamic Memory **
 <DM> OPEN,  Ldi:  1, File: KP.DBR , Attr: old, Block I/O
SET ECHO: ON
SET FILENAME: ka.log
SET LDIR:    1
SET IDSN:    1
SET DATASET_NAME: ARCHIVE.SUMMARY
SET BEG_MESH:   1
SET END_MESH:   2
SET INC_MESH:   1
```

```
=======>  Processing mesh   0
=======>  Processing mesh   1

          Total CPU time recorded          31.000000000

          CPU summary
              ARC              0.000000000       0.00
              ES               9.700000000       0.31
              RENUMB           1.000000000       0.03
              COP              1.600000000       0.05
              ASM              5.100000000       0.16
              FAC,SOL          1.100000000       0.04
              ERR              4.200000000       0.14
              REF              8.300000000       0.27
              VEC              0.000000000       0.00
              MISC             0.000000000       0.00
           ----------------------------------------
              TOTAL           31.000000000       1.00

                  =======>  Processing mesh   2

                Total CPU time recorded        109.800000000

                                CPU summary
                           ARC              0.000000000       0.00
                           ES              20.300000000       0.26
                           RENUMB           1.700000000       0.02
                           COP              3.800000000       0.05
                           ASM             19.600000000       0.25
                           FAC,SOL         10.500000000       0.13
                           ERR              7.100000000       0.09
                           REF             15.800000000       0.20
                           VEC              0.000000000       0.00
                           MISC             0.000000000       0.00
                         ----------------------------------------
                            TOTAL          78.800000000       1.00

                         =======>  Processing mesh   3

      EXIT PST      CPUTIME=       0.72  I/O(DIR,BUF)=       0       0
                           <CL> PNS exhausted
                            ENDRUN called by CLIP
                         <DM> CLOSE, Ldi:  1, File: KP.DBR
```

14.4.2.4 DISPLACEMENT Command

14.4.2.4.1 General Description

This command is similar to the VECTOR command, except that the NVT vector is automatically assumed to be the NODAL.DISPLACEMENT vector.

14.4.2.4.2 Command Syntax

The command syntax for the DISPLACEMNT command is shown below.

```
ARCHIVE
  SET MESH = beg_mesh, end_mesh, inc_mesh
  DISPLACEMENT

     [ABS  { <true> | <false> }]
     {{MIN | MAX} { <true> | <false> } ]
     [INPUT_NODES  node-list_record_name]

     [OUTPUT_NODE  maxmin-node_record_name]
     [OUTPUT_COORD maxmin-coordinate_record_name]
     [OUTPUT_DISP  disp_record_name]
     [OUTPUT_COMP  maxmin-node_record_name]
     ARCHIVE       [INPUT_NODES node-list_record_name]
  RETURN
```

The above command syntax assumes that the default name of the node list record group is used.

14.4.2.5 NODES Command

14.4.2.5.1 General Description

The NODES command generates a list of nodes typically used to extract nodal displacements, stress, strain, or strain energy quantities from the database. The node list is created either by directly specifying the nodes to be included in the node list or by specifying a coordinate range and a tolerance, used to search for nodes in the NCT data object that fall within the coordinate range specified. The node list is then usually passed to the DISPLACEMENT, VECTOR, or RESULTANT command to extract resultant solution quantities for the nodes in the node list.

14.4.2.5.2 Command Syntax

The command syntax for the NODES command is shown below.

```
ARCHIVE
   SET MESH = beg_mesh, end_mesh, inc_mesh
   NODES
        [OUTPUT_NODE  node-list_record_name]
        [OUTPUT_COORD coordinate_record_name]
        NODE  beg_node, end_node, inc_node
        ARCHIVE
   RETURN
STOP
```

The NODES function is invoked by first executing the ARCHIVE sub-processor and then invoking the NODES command from the ARCHIVE sub-processor. The SET commands are used to augment the NODES sub-commands shown below. The actual formation of the node list is initiated by entering the ARCHIVE command.

Table 14.4-14 Sub-command Names for the NODES Command

Command Name	Default	Function
ARCHIVE		Archives the nodes specified via the NODES command.
NODES	ALL	Command used to specify the node list. If this sub-command is specified without any other parameters all active nodes for the mesh range specified are extracted.
OUTPUT_NODE	NODE	Name of record group where the output nodes are stored.
OUTPUT_COORD	CORD	Name of record group where the coordinates of the selected nodes are stored.
RETURN		Return from ARCHIVE sub-processor to main executive level.

14.4.2.5.3 The ARCHIVE Command

This command initiates the archival process.

```
ARCHIVE
```

14.4.2.5.4 The NODE Command

This command specifies the nodes to be included in a node list. The nodes are either specified directly by listing the nodes to be included in the node list or indirectly by specifying a coordinate range from which the nodes should be extracted. If the nodes are specified directly, the starting node (*beg_node*), ending node (*end_node*), and the node increment (*inc_node*) of the nodes to be included are supplied as input. When this format is used, the command can be repeated multiple times before the ARCHIVE command is issued to initiate the archival process. All nodes specified in this way will be added to the node list.

```
NODE  beg_node, end_node, inc_node
```

Alternatively the nodes can be extracted by specifying a coordinate range. In this mode, the default is to assume that all nodes are included in the node list. The user then proceeds to limit the number of nodes to be included in the final node-list by specifying a coordinate range from which the nodes should be extracted. All nodes which fall within the coordinate range (to within a specified tolerance) are extracted and entered into a GAL record.

```
NODE  X=x_beg,x_end Y=y_beg,y_end Z=z_beg,z_end MACRO=name
```

Table 14.4-15 List of Parameters for NODE Command

Parameter	Function
beg_node	Beginning node number in loop limit node specification
end_node	Ending node number in loop limit node specification

Table 14.4-15 List of Parameters for NODE Command (Continued)

Parameter	Function
inc_node	Increment in loop limit node specification
x_{beg}	Beginning x coordinate in range specification
x_{end}	Ending x coordinate in range specification
y_{beg}	Beginning y coordinate in range specification
y_{end}	Ending y coordinate in range specification
z_{beg}	Beginning z coordinate in range specification
z_{end}	Ending z coordinate in range specification
name	Name of macrosymbol containing node list. Each node in node list is contained in an indexed macrosymbol. The number of items in the macro array is stored in a macrosymbol called *num_name*.

One or all of the keywords X, Y, Z, or MACRO may be included. If none are specified, all active nodes in the NCT data object are included.

14.4.2.5.5 The OUTPUT_NODE Command

This command sets the name of the record group containing the node list. If the node list is generated by using a coordinate range specification, the number of nodes extracted for each mesh must be the same, since all records in a record group must have the same length (fixed-length records). If you expect that based on the coordinate range specification a different number of nodes will be extracted for each mesh, then each mesh should be processed separately and a unique record name should be assigned for each mesh. If the NODES command is invoked and the SET MESH command is such that *beg_mesh* = *end_mesh* then the record cycle number is always one, irrespective of mesh number. If *beg_mesh* < *end_mesh* then a record group is created where the record cycle number corresponds to *mesh+1*.

```
OUTPUT_NODE node_list_record_name
```

The parameter *node_list_record_name* is the name of the record group and is limited to 11 characters. The default name is NODE. If the SET TAG command is issued the tag string gets appended to the default name. The name of the NODE record group is automatically passed to the VECTOR, DISPLACEMENT, or RESULTANT command if the default name or the TAG name is used. If the name of the node list is supplied via the OUTPUT_NODE command, then you have to pass this name explicitly to the other ARCHIVE functions.

14.4.2.5.6 The OUTPUT_COORD Command

This command sets the name of the record group containing the coordinates of the nodes contained in the node list. The record group created is a fixed-length record group and the cycle number appended to the record name follows the same convention as for the OUTPUT_NODE command above.

```
OUTPUT_COORD  coordinate_record_name
```

The parameter *coordinate_record_name* is the name of the record group and is limited to 11 characters. The default name is CORD. If the SET TAG command is issued the tag string gets appended to the default name.

14.4.2.5.7 The RETURN Command

This command exits from NODES command level and returns to the ARCHIVE command level.

```
RETURN
```

14.4.2.5.8 Relevant SET Commands

Table 14.4-16 lists all SET commands relevant to the CPU command.

Table 14.4-16 SET Commands Relevant to NODES Command

Command Name	Default	Function
SET CONSTRAINT_SET	1	Constraint set
SET DATASET_NAME	1,ARCHIVE	Name of dataset which receives the archived quantities as a record group.
SET LDI	1	Logical device index of central data library (.DBC).
SET LDI/ARCHIVE	1	Logical device index for results data library (.DBR).
SET MESH	0,0,1	Range of meshes from which data is to be archived.
SET TAG	-	Tag that is appended to the default record names in order to create unique record names. The tag is limited to 6 characters.
SET TOL	0.001	Tolerance used for identifying nodes at certain coordinate locations.

14.4.2.5.9 Input Datasets

This command searches the nodal coordinate table (NCT) for active nodes and therefore only requires the CSM, NCT, and NDT data objects. The CSM is required for general model and element type attributes, the NDT is required to check which nodes are active, and the NCT stores the nodal coordinates.

Table 14.4-17 NODES Command Input Datasets

Dataset	Type	Description
CSM.SUMMARY...*mesh*	CSM	Complete Summary of the Model (CSM data object)
NODAL.COORDINATE..*mesh*	NAT	Nodal Attribute Table (NAT) of pseudo-vectors used to update nodal rotation triads. The name of an NAT data object is completely specified by the user, including component names and cycle numbers.

Table 14.4-17 NODES Command Input Datasets (Continued)

Dataset	Type	Description
NODAL.DOF..*con_set.mesh*	NDT	Nodal DOF Table (NDT). The name of an NDT data object is completely specified by the user, including component names and cycle numbers

14.4.2.5.10 Output Datasets

The output from this command is written to the resultant data library (DBR file) in the form of record groups. The dataset name of the record group is set with the SET DATASET command. The names of the records created are controlled by the SET TAG command or the OUTPUT_NODE and OUTPUT_COORD command. Alternatively, the user can also use the default values shown in Table 14.4-18 or Table 14.4-19.

Table 14.4-18 NODES Command Output Datasets

Dataset Name	Default	Description
dataset_name	1,ARCHIVE	Name of archival dataset name

The name of the output record names can be constructed in two different ways. The user can either explicitly supply the name of the record group, or supply a tag string which is appended to the default name. The tag string is set with the SET TAG command.

Table 14.4-19 NODES Command Output Records

Record Name	Default	Description
NODE*tag or output_node*	NODE	Node number at which maximum value is located
CORD*tag or output_coordinate*	CORD	Nodal component number at which maximum value is located

14.4.2.5.11 Limitations

None.

14.4.2.5.12 Error Messages

The error messages produced by this command are listed in Table 14.4-20.

Table 14.4-20 Error Messages Produced by the NODES Command

Error #	Error Message	Probable Cause(s)	Recommended User Response
1	Incorrect number of input items ...	Command line contains too many or too few input parameters	Re-enter command with correct number of parameters. Refer to PST documentation if necessary.

Table 14.4-20 Error Messages Produced by the NODES Command (Continued)

Error #	Error Message	Probable Cause(s)	Recommended User Response
2	Node number specified must be greater than zero.	Invalid node range.	Re-enter node range in loop-limit format with positive integer numbers.
3	Unable to assign/resize space for ???? buffer.	Error in node list input, otherwise internal variables have been clobbered. It is possible the system is out of memory.	Verify node list input. If the system is out of memory, release memory for process. Read complete error diagnostic. If meaning unclear see developer.
4	Command not recognized	Command issued is misspelled or not applicable	Supply valid command.
5	Cannot open CSM/NCT/ NDT data object...	Data object does not exist for mesh number or logical device index specified.	Check data library and set correct mesh number and logical device index.
6	Cannot read NCT/NDT data object	Clobbered database.	Read complete error diagnostic. If meaning unclear see developer.
7	Unable to find a node within the coordinate range and tolerance specified.	Search for node is not satisfied because tolerance specified is too small. Coordinate range specified does not contain any nodes.	Reset tolerance and verify coordinate range.
8	Problems opening the ???? record group.	The resultant data library is not open or the name of the results dataset is invalid. The record group exists and was previously opened with a different fixed-length record size.	Open resultant data library with *OPEN directive. Specify a valid dataset name. Specify a different record group name
9	Unable to write ???? record.	Bad record group identifier, internal tables clobbered.	Read complete error diagnostic. If meaning unclear see developer.
10	Unable to close CSM object	Bad HDB data object handle of data library conflict.	Read complete error diagnostic. If meaning unclear see developer.
11	Problem closing ???? record group	Bad DB record group identifier, or dataset clobbered.	Read complete error diagnostic. If meaning unclear see developer.

14.4.2.5.13 Examples and Usage Guidelines

This first input example illustrates how to use the NODES command to generate a node list by directly specifying the nodes in the list. The NODES command will include the nodes specified in the node list and retrieve the coordinates of these nodes. The nodes entered here describe the radius of the central hole of the Knight's Panel problem for computational mesh number one.

```
   *echo,off

   *open KP.DBC
   *open KP.DBR

.  Set General Parameters
.  ======================
   SET ECHO        = <true>
   SET LDI         = 1
   SET DATA        = 2, ARCHIVE
   SET MESH        = 1


   ARCHIVE

.     Define Node to extract displacements from
.     =========================================
      NODES
         Node   17,24,1
         Node   29,30,1
         Node   45,46,1
         Node   59,60,1
         Node   71,72,1
         ARCHIVE

      RETURN

.  Display content of results library
.  ==================================
   *rat 2
   *print 2 1

   STOP
```

The output record group NODE includes all the nodes specified as input, and the output record group CORD lists all the coordinates of the nodes in the node list. Here the SET ECHO command was used to echo all the SET commands to standard output after they have been issued.

```
 <CL> PUT_message,Commnt>
 ** BEGIN PST       ** Using Dynamic Memory **
 <DM> OPEN,  Ldi:  1, File: KP.DBC , Attr: old, Block I/O
 <DM> OPEN,  Ldi:  2, File: KP.DBR , Attr: new, Block I/O
SET ECHO: ON
SET LDI:   1
SET LDIR:    2
SET IDSN:    0
SET DATASET_NAME: ARCHIVE
SET MESH:   1
```

```
Record Table of dataset ARCHIVE
Key             L_cyc  H_cyc  Type Log_size
CORD                1      1   D         48
NODE                1      1   I         16

 Record CORD.1 of dataset ARCHIVE
  1: -1.4142D+00  1.4933D+01  8.4142D+00 -6.5567D-07  1.5000D+01
     9.0000D+00  1.4142D+00  1.4933D+01  8.4142D+00  2.0000D+00
 11:  1.4866D+01  7.0000D+00  1.4142D+00  1.4933D+01  5.5858D+00
    -6.5567D-07  1.5000D+01  5.0000D+00 -1.4142D+00  1.4933D+01
 21:  5.5858D+00 -2.0000D+00  1.4866D+01  7.0000D+00 -7.0711D-01
     1.4983D+01  8.8536D+00  7.0711D-01  1.4983D+01  8.8536D+00
 31:  1.8536D+00  1.4883D+01  7.7071D+00  1.8536D+00  1.4883D+01
     6.2929D+00  7.0711D-01  1.4983D+01  5.1464D+00 -7.0711D-01
 41:  1.4983D+01  5.1464D+00 -1.8536D+00  1.4883D+01  6.2929D+00
    -1.8536D+00  1.4883D+01  7.7071D+00

 Record NODE.1 of dataset ARCHIVE
  1:          17          18          19          20          21
              22          23          24          29          30
 11:          45          46          59          60          71
              72

EXIT PST      CPUTIME=      0.66  I/O(DIR,BUF)=      0      0
<CL> PNS exhausted
     ENDRUN called by CLIP
<DM> CLOSE, Ldi:  1, File: KP.DBC
<DM> CLOSE, Ldi:  2, File: KP.DBR
```

The example shown above is an example of a NODES specification where a coordinate range is specified. In the next example, all nodes situated in the xy-plane with z_g=0.0, are archived. This format of the NODES command lets the user specify a coordinate range and tolerance from which the nodes in the node list are selected. Note that the user does not have to specify a coordinate range for all coordinate directions. Without a coordinate range specification all nodes are included; each coordinate range specified restricts the number of nodes to include in the final node list.

This example also shows how you can control the name of the output record groups archiving the node list and coordinate list. Since *beg_mesh* = *end_mesh*, only one record was created with cycle number one.

```
   *echo,off

   *open KP.DBC
   *open KP.DBR

.  Set General Parameters
.  ======================
   SET LDI         = 1
   SET DATA        = 2, ARCHIVE
   SET MESH        = 2
```

```
ARCHIVE

.    Define Node to extract displacements from
.    ========================================
     NODES
        NODE  Z=0.0   MACRO = Z_0
        OUTPUT_NODE = SPECIF_NODE
        OUTPUT_COORD= SPECIF_COORD
        ARCHIVE

     RETURN

.  Display macrosymbols created
.  ============================
   *show macros

.  Display content of results library
.  ==================================
   *rat 2

   STOP
```

The following shows the output in response to the input shown above. In addition to creating record groups, macrosymbols are created.

```
<CL> PUT_message,Commnt>
** BEGIN PST       ** Using Dynamic Memory **
<DM> OPEN,  Ldi:  1, File: KP.DBC , Attr: old, Block I/O
<DM> OPEN,  Ldi:  2, File: KP.DBR , Attr: old, Block I/O

<CL> lv:0 t:I       CSM_PRECISION  2
<CL> lv:0 t:I       Z_0[1]         5
<CL> lv:0 t:I       Z_0[2]         6
<CL> lv:0 t:I       Z_0[3]         7
<CL> lv:0 t:I       Z_0[4]         55
<CL> lv:0 t:I       Z_0[5]         56
<CL> lv:0 t:I       NUM_Z_0        5

Record Table of dataset ARCHIVE
Key           L_cyc  H_cyc  Type Log_size
SPECIF_COOR       1      1   D         15
SPECIF_NODE       1      1   I          5

EXIT PST      CPUTIME=      0.90  I/O(DIR,BUF)=      0      0
<CL> PNS exhausted
     ENDRUN called by CLIP
<DM> CLOSE, Ldi:  1, File: KP.DBC
<DM> CLOSE, Ldi:  2, File: KP.DBR
```

The final example specifies a coordinate location where the user wants to view the value of resultant data. The program will search for nodes, within a specified tolerance of this location, and archive those in the data library. Since only one value is extracted for each mesh (the one closest to the point specified) a mesh range can be specified for which this function is performed. (The output can be stored in a fixed-length record group.) This example also shows how the SET TAG command is used. It must be issued from within the ARCHIVE sub-processor, otherwise the TAG is reset to its default value.

```
   *echo,off

   *open KP.DBC
   *open KP.DBR

.  Set General Parameters
.  ======================
   SET ECHO         = <true>
   SET LDI          = 1
   SET DATA         = 2, ARCHIVE
   SET MESH         = 0,4,1
   SET TOL          = 0.01

   ARCHIVE

      SET TAG          = MAXVAL

.     Define Node to extract displacements from
.     =========================================
      NODES
         NODE  X=2.0  Y=14.866  Z=7.0
         ARCHIVE

      RETURN

.  Display content of results library
.  ==================================
   *rat 2
   *print 2 1

   STOP
```

Below is the output in response to the input shown above.

```
<CL> PUT_message,Commnt>
** BEGIN PST       ** Using Dynamic Memory **
<DM> OPEN,  Ldi:  1, File: KP.DBC , Attr: old, Block I/O
<DM> OPEN,  Ldi:  2, File: KP.DBR , Attr: old, Block I/O
Record Table of dataset ARCHIVE
Key           L_cyc  H_cyc  Type Log_size
CORDMAXVAL        1      5   D          3
NODEMAXVAL        1      5   I          1

 Record CORDMAXVAL.1 of dataset ARCHIVE
 1:   2.0000D+00  1.4866D+01  7.0000D+00
 Record NODEMAXVAL.1 of dataset ARCHIVE
 1:           20
 Record SPECIF_COOR.1 of dataset ARCHIVE
  1:  7.0000D+00  1.3266D+01  0.0000D+00 -6.5567D-07  1.5000D+01
      0.0000D+00 -7.0000D+00  1.3266D+01  0.0000D+00  3.5000D+00
 11:  1.4567D+01  0.0000D+00 -3.5000D+00  1.4567D+01  0.0000D+00
```

```
 Record SPECIF_NODE.1 of dataset ARCHIVE
 1:            5           6           7          55          56
 Record CORDMAXVAL.2 of dataset ARCHIVE
 1:   2.0000D+00  1.4866D+01  7.0000D+00
 Record NODEMAXVAL.2 of dataset ARCHIVE
 1:           20
 Record CORDMAXVAL.3 of dataset ARCHIVE
 1:   2.0000D+00  1.4866D+01  7.0000D+00
 Record NODEMAXVAL.3 of dataset ARCHIVE
 1:           20
 Record CORDMAXVAL.4 of dataset ARCHIVE
 1:   2.0000D+00  1.4866D+01  7.0000D+00
 Record NODEMAXVAL.4 of dataset ARCHIVE
 1:           20
 Record CORDMAXVAL.5 of dataset ARCHIVE
 1:   2.0000D+00  1.4866D+01  7.0000D+00
 Record NODEMAXVAL.5 of dataset ARCHIVE
 1:           20
EXIT PST      CPUTIME=      0.90  I/O(DIR,BUF)=      0       0
<CL> PNS exhausted
     ENDRUN called by CLIP
<DM> CLOSE, Ldi:  1, File: KP.DBC
<DM> CLOSE, Ldi:  2, File: KP.DBR
```

14.4.2.6 RESULTANT Command

14.4.2.6.1 General Description

The RESULTANT command extracts nodally smoothed stress, strain, and strain energy quantities from the database. There are two modes of operation: either the user specifies a node list for which nodal resultant quantities are extracted, or the database is searched for the node at which the stress based nodal resultant is a maximum or minimum.

14.4.2.6.2 Command Syntax

The command syntax for the RESULTANT command is shown below.

```
ARCHIVE

  RESULTANT
     [ABS  { <true> | <false> }]
     [BARLOW { <true> | <false> }]
     {(MIN | MAX) { <true> | <false> } ]
     [NUM_GROUPS = num_group, (grp1,grp2,...,grpnum_group)]
     [INPUT_NODES   node-list_record_name]

     [OUTPUT_COMP   maxmin-comp_record_name]
     [OUTPUT_NODES  maxmin-node_record_name]
     [OUTPUT_STRESS stress_record_name]
     {STRESS | STRAIN | STRAIN_ENERGY }
    ARCHIVE      [INPUT_NODES node-list_record_name]
  RETURN
```

The commands associated with the RESULTANT function are described in Table 14.4-21. All these commands are optional except for the {STRESS | STRAIN | STRAIN_ENERGY} command. Specify the results type to archive by selecting one of the keywords STRESS, STRAIN, or STRAIN_ENERGY. To initiate the archival enter the ARCHIVE command.

Table 14.4-21 Summary of RESULTANT Commands

Command Name	Default	Function
ABS	0	Use absolute value of components when determining the extremum value.
ARCHIVE		Initiate archival of resultant components from database.
BARLOW	<false>	Extrapolate to nodes via Barlow points.
INPUT_NODES	NODE	Name of the record group containing the input node list.
MAX	<false>	This command searches the results object for the component with the maximum value. When used in conjunction with the ABS flag, the search is performed on the absolute value of the components.
MIN	<false>	This command searches the results object for the component with the minimum value. When used in conjunction with the ABS flag, the search is performed on the absolute value of the components.
NUM_GROUPS	1,1	Specifies the element groups to be include in the computation of smooth stress, strain, or strain energy fields.
OUTPUT_COMP	COMP	Name of the record group containing the component of the node at which the maximum stress value is located. This record group is only created if the MAX or MIN command is specified.
OUTPUT_NODE	NODR	Name of the record group containing the node number of the node at which the maximum stress value is located. This record group is only created if the MAX or MIN command is specified.

Table 14.4-21 Summary of RESULTANT Commands (Continued)

Command Name	Default	Function
OUTPUT_STRESS	SIGM	Name of the record group containing the value(s) of the stress component extracted.
	EPSI	Name of the record group containing the value(s) of the strain component extracted.
	UFE	Name of the record group containing the value(s) of the strain energy density extracted.
RETURN		Return to ARCHIVE main command level.
STRAIN	<false>	Flag indicating that strain components are to be extracted and archived.
STRAIN_ENERGY	<false>	Flag indicating that the strain energy density is to be extracted and archived.
STRESS	<false>	Flag indicating that stress components are to be extracted and archived.

14.4.2.6.3 The ABS Command

This command indicates that the absolute value of the nodal results should be used when searching for an extremum value.

```
ABS  { <true> | <false> }
```

14.4.2.6.4 The ARCHIVE Command

This command initiates the archival process.

```
ARCHIVE
```

14.4.2.6.5 The BARLOW Command

This command allows you to control when stress, strain, and strain energy components are extrapolated from integration points to nodes via Barlow points.

```
BARLOW [ <true> | <false> ]
```

14.4.2.6.6 The INPUT_NODE Command

This command specifies the name of the record group which contains the input node list. It is only used if the MAX or MIN command is not specified.

```
INPUT_NODE node_list_record_name
```

The *node_list_record_name* is the name of the record group and is limited to 11 characters. The default name is NODE. If the SET TAG command is issued the tag string gets appended to the default name.

14.4.2.6.7 The MAX Command

Search the results quantity selected for the component with the maximum value.

```
MAX [ <true> | <false> ]
```

14.4.2.6.8 The MIN Command

Search the results quantity selected for the component with the minimum value.

```
MIN [ <true> | <false> ]
```

14.4.2.6.9 The NUM_GROUPS Command

This command selects the element groups to be included when smoothing stress, strain, or strain energy density.

NUM_GROUP *num_group*, ($grp_1, grp_2, \ldots, grp_{num_group}$)

14.4.2.6.10 The OUTPUT_COMP Command

Name of record group containing the nodal component for which the extremum value was found. This record group is only created if the MAX or MIN command was used.

```
OUTPUT_COMP maxmin_comp_record_name
```

The *maxmin_comp_record_name* is the name of the record group and is limited to 11 characters. The default name is COMP. If the SET TAG command is issued the tag string gets appended to the default name.

14.4.2.6.11 The OUTPUT_NODE Command

Name of record group containing the node number for which the extremum value was found. This record group is only created if the MAX or MIN command was used.

```
OUTPUT_NODE maxmin_node_record_name
```

The *maxmin_node_record_name* is the name of the record group and is limited to 11 characters. The default name is NODR. If the SET TAG command is issued the tag string gets appended to the default name.

14.4.2.6.12 The OUTPUT_STRESS Command

Name of record group containing archived stress component. This record group is always created.

```
OUTPUT_STRESS stress_record_name
```

The *stress_record_name* is the name of the record group and is limited to 11 characters. The default name is SIGM, EPSI, or UFE, depending on the resultant type chosen. If the SET TAG command is issued the tag string is appended to the default name

14.4.2.6.13 The RETURN Command

This command returns program control to the ARCHIVE sub-processor command level.

```
RETURN
```

14.4.2.6.14 The STRESS Command

Flag indicating that stresses are to be extracted and archived. When this command is issued it supersedes any previous STRESS, STRAIN, or STRAIN_ENERGY command. This command automatically sets the output stress record group to its default value (SIGM).

```
STRESS [ <true> | <false> ]
```

14.4.2.6.15 The STRAIN Command

Flag indicating that strains are to be extracted and archived. When this command is issued it supersedes any previous STRESS, STRAIN, or STRAIN_ENERGY command. This command automatically sets the output stress record group to its default value (EPSI).

```
STRAIN [ <true> | <false> ]
```

14.4.2.6.16 The STRAIN_ENERGY Command

Flag indicating that strain energy densities are to be extracted and archived. When this command is issued it supersedes any previous STRESS, STRAIN, or STRAIN_ENERGY command. This command automatically sets the output stress record group to its default value (UFE).

```
STRAIN_ENERGY [ <true> | <false> ]
```

14.4.2.6.17 Relevant SET Commands

Table 14.4-22 lists all SET commands relevant to the RESULTANT command.

Table 14.4-22 SET Commands Relevant to RESULTANT Command

Command Name	Default	Function
SET COMPONENT	0	Stress component selected: 0 — All resultant stress components at a node are included in the archival function >1 — Only the component selected via this command is included in the archival function
SET CONSTRAINT_SET	1	Constraint set to use.
SET DATASET_NAME	1,ARCHIVE	Name of dataset which receives archived quantities as a record group.
SET LDI	1	Logical device index of central data library (.DBC).
SET LDI/ARCHIVE	1	Logical device index for results data library (.DBR).
SET LOAD_SET	1	Load Set.
SET MESH	0,0,1	Range of meshes from which data is to be archived.
SET TAG	-	This is a tag that is appended to the default record names in order to create unique record names. The tag is limited to 6 characters.

14.4.2.6.18 Input Datasets

Table 14.4-23 RESULTANT Command Input Dataset Names

Dataset	Type	Description
CSM.SUMMARY...*mesh*	CSM	Complete Summary of the Model (CSM data object)
NODAL.DISPLACEMENT. *load_set.con_set.mesh*	NAT	Nodal Attribute Table (NAT) of pseudo-vectors used to update nodal rotation triads. The name of an NAT data object is completely specified by the user, including component names and cycle numbers.
NODAL.DO*F..con_set.mesh*	NDT	Nodal DOF Table (NDT). The name of an NDT data object is completely specified by the user, including component names and cycle numbers
NODAL.TRANSFORMATION...*mesh*	NTT	Nodal vector Table (NVT). The name of an NVT data object is completely specified by the user, including component names and cycle numbers.
eltnam.STRESS.*load_set.con_set.mesh*	EST	Element stress Table (EST). The element stresses are stored at integration points.
eltnam.STRAIN.*load_set.con_set.mesh*	EST	Element stress Table (EST). The element strains are stored at integration points.
eltnam.STRAIN_ENERGY. *load_set.con_set.mesh*	EST	Element stress Table (EST). The element strain energy densities are stored at integration points.

14.4.2.6.19 Output Datasets

Table 14.4-24 RESULTANT Command Output Dataset Names

Dataset Name	Default	Description
dataset_name	1, ARCHIVE	Name of archival dataset name

If the input data is a node list then the output from the RESULTANT function is a list of stress resultants corresponding to the nodes in the node list. Stresses are only extracted for stress components selected via the SET COMP command.

Table 14.4-25 RESULTANT Command Output Record Names

Record Name	Description
NODR	Node number at which maximum value is located.
COMP	Nodal component number at which maximum value is located
SIGM	Name of record group containing archived stresses.
EPSI	Name of record group containing archived strains.
UFE	Name of record group containing archived strain energy densities.

The default output record name can be modified by the SET TAG command or by directly specifying alternate record names via the OUTPUT_STRESS command.

14.4.2.6.20 Limitations

None.

14.4.2.6.21 Error Messages

Most messages produced are self-explanatory.

14.4.2.6.22 Examples and Usage Guidelines

The following is an example of an input deck where the NODES and RESULTANT command have been used together to archive stresses, strains, and strain energy densities.

```
*open KP.DBC
*open KP.DBR

. Set General Parameters
. ======================
SET DATA        = 2, ARCHIVE.SUMMARY
SET MESH        = 0,4,1
SET COMP        = 1

ARCHIVE

.    Define Node to extract displacements from
.    =========================================
     SET TAG         = MAXVAL
     SET TOL         = 0.01

     NODES
        NODE  X=2.0  Y=14.866  Z=7.0
        ARCHIVE

     RESULTANT
        BARLOW = <true>
        STRAIN
        ARCHIVE

     RESULTANT
        BARLOW = <true>
        STRESS
        ARCHIVE

     RESULTANT
        BARLOW = <true>
        STRAIN_ENERGY
        ARCHIVE

     RETURN

. Display content of results library
. ===================================
*rat 2
*print 2 1

STOP
```

14.4.2.7 VECTOR Command

14.4.2.7.1 General Description

The VECTOR command extracts selected nodal data from a vector stored either as an NVT or an NAT object. The node list is selected via the NODES command as described in Section 14.4.2.5. The output from this command depends on the additional input parameters specified to the VECTOR command. If the keyword MAX or MIN is selected, then the stress components of the nodes in the node list are searched for a maximum or minimum value, which is written to the

archival data library. If neither the MAX nor the MIN keyword is specified, then all stress components of the nodes in the node list are extracted and written to the archival data library. If the NODES command is omitted, then all nodes (active and inactive) are considered part of the node list and are written to the archival data library.

14.4.2.7.2 Command Syntax

The command syntax for the VECTOR command is shown below.

```
ARCHIVE
  SET MESH = beg_mesh, end_mesh, inc_mesh
  VECTOR
     {NVT_NAME | NAT_NAME }= dsname

     [ABS  { <true> | <false> }]
     {{MIN | MAX} { <true> | <false> } ]
     [INPUT_NODES  node-list_record_name]

     [OUTPUT_NODE  maxmin-node_record_name]
     [OUTPUT_COORD maxmin-coordinate_record_name]
     [OUTPUT_DISP  disp_record_name]
     [OUTPUT_COMP  maxmin-node_record_name]
     ARCHIVE       [INPUT_NODES node-list_record_name]
  RETURN
```

Table 14.4-26 Summary of VECTOR Commands

Command Name	Default	Function
ABS	<false>	Use absolute value of components when determining the extremum value.
ARCHIVE	-	Archive vector components from database.
INPUT_NODES	NODE	Name of the record group containing the input node list.
MAX	<false>	This command searches the vector for the component with the maximum value. When used in conjunction with the ABS flag, the search is performed on the absolute value of the components.
MIN	<false>	This command searches the vector for the component with the minimum value. When used in conjunction with the ABS flag, the search is performed on the absolute value of the components.
NAT_NAME	NODAL.DIS-PLACEMENT	Name of NAT vector used for archival purposes.
NVT_NAME	NODAL.ROTA-TION	Name of NVT vector used for archival purposes.
OUTPUT_COMP	COMP	Name of the record group containing the component of the node at which the maximum vector value is located. This record group is only created if the MAX or MIN command is specified.

Table 14.4-26 Summary of VECTOR Commands (Continued)

Command Name	Default	Function
OUTPUT_NODE	NODD	Name of the record group containing the node number of the node at which the maximum vector value is located. This record group is only created if the MAX or MIN command is specified.
OUTPUT_COORD	CORD	Name of the record group containing the coordinate of the node at which the maximum vector value is located. This record group is only created if the MAX or MIN command is specified.
OUTPUT_DISP	DISP	Name of record group containing the displacement value at the nodes selected.
RETURN		Return to ARCHIVE command module.

14.4.2.7.3 The ABS Command

This command indicates that the absolute value of the nodal vector components should be used when searching for the extremum value.

```
ABS  { <true> | <false> }
```

14.4.2.7.4 The ARCHIVE Command

This command initiates the archival process.

```
ARCHIVE
```

14.4.2.7.5 The INPUT_NODE Command

This command specifies the name of the record group that contains the input node list. If the MAX or MIN command is selected this command is redundant.

```
INPUT_NODE node_list_record_name
```

The *node_list_record_name* is the name of the record group and is limited to 11 characters. The default name is NODE. If the SET TAG command is issued the tag string is appended to the default name.

14.4.2.7.6 The MAX Command

This command indicates that the input vector should be searched for the maximum vector component value.

```
MAX  { <true> | <false> }
```

14.4.2.7.7 The MIN Command

This command indicates that the input vector should be searched for the minimum vector component value.

```
MIN   { <true> | <false> }
```

14.4.2.7.8 The OUTPUT_COMP Command

Name of record group containing the nodal component for which an extremum value was found. This record group is only created if the MAX or MIN command was used.

```
OUTPUT_COMP maxmin_comp_record_name
```

The *maxmin_comp_record_name* is the name of the record group and is limited to 11 characters. The default name is COMP. If the SET TAG command is issued the tag string is appended to the default name.

14.4.2.7.9 The OUTPUT_COORD command

Name of the record group containing the coordinate of the node at which the maximum vector value is located. This record group is only created if the MAX or MIN command is specified.

```
OUTPUT_COORD  coordinate_record_name
```

The *coordinate_record_name* is the name of the record group and is limited to 11 characters. The default name is CORD. If the SET TAG command is issued the tag string is appended to the default name.

14.4.2.7.10 The OUTPUT_DISP Command

Name of record group containing the displacement value of the node(s) archived.

```
OUTPUT_DISP  disp_record_name
```

The *disp_record_name* is the name of the record group and is limited to 11 characters. The default name is DISP. If the SET TAG command is issued the tag string is appended to the default name.

14.4.2.7.11 The OUTPUT_NODE Command

Name of record group containing the node for which an extremum value was found. This record group is only created if the MAX or MIN command was used.

```
OUTPUT_NODE maxmin_node_record_name
```

The *maxmin_node_record_name* is the name of the record group and is limited to 11 characters. The default name is NODD. If the SET TAG command is issued the tag string is appended to the default name.

14.4.2.7.12 The RETURN Command

This command exits from the NODES command level and returns to ARCHIVE command level.

```
RETURN
```

14.4.2.7.13 Relevant SET Commands

Table 14.4-27 summarizes all SET commands relevant to the VECTOR command.

Table 14.4-27 SET Commands Relevant to VECTOR Command

Command Name	Default	Function
SET LDI	1	Logical device index of computational data library.
SET LDI/ARCHIVE	1	Logical device index for results data library.
SET DATASET_NAME	1,ARCHIVE	Name of dataset which will contain CPU record group.
SET MESH	0	Adaptive mesh number.
SET FRAME	GLOBAL	Coordinate frame in which the displacement components are viewed. GLOBAL – Transform vector to global frame. COMPUTATIONAL – Transform vector to computational frame.
SET COMPONENT	0	Selects nodal component: = 0 All components > 1 Component number

14.4.2.7.14 Input Datasets

Table 14.4-28 Processor PST Input Datasets

Dataset	Type	Description
CSM.SUMMARY...*mesh*	CSM	Complete Summary of the Model (CSM data object)
NODAL.DOF..*con_set.mesh*	NDT	Nodal DOF Table (NDT). The name of an NDT data object is completely specified by the user, including component names and cycle numbers
NODAL.TRANSFORMATION...*mesh*	NTT	Nodal vector Table (NVT). The name of an NVT data object is completely specified by the user, including component names and cycle numbers.
NVT_*name.load_set.con_set.mesh*	NVT	Name of nodal vector stored as an NVT object. The user specifies the name of the NVT object.
NAT_*name.load_set.con_set.mesh*	NAT	Name of nodal vector stored as an NAT object. The user specifies the name of the NAT object.

14.4.2.7.15 Output Datasets

The output from the VECTOR command is a set of GAL record groups. The name and content of these record groups is described in Table 14.4-29.

Table 14.4-29 VECTOR Command Output Datasets

Record Name	Description
NODD	Node number at which maximum value is located.
COMP	Nodal component number at which maximum value is located.
DISP	Maximum component value whose location is described by NODD and COMP.

If the input data is in the form of a node list then the output of this operation is a list of output displacements corresponding to the node list and the component selected. The default output record name is DISP but can be modified by the SET TAG command or by directly specifying an alternate record name with the OUTPUT_DISP command as described above.

14.4.2.7.16 Limitations

None.

14.4.2.7.17 Error Messages

Most messages produced are self-explanatory.

14.4.2.7.18 Examples and Usage Guidelines

In the following example the user specifies a location in the form of a node number at which to extract displacement data. The Z component at node 442 in the global frame is extracted and put into the record group DISP442. This operation is performed for Mesh=0 through Mesh=4.

```
*open 1 HSCT.DBC
*open 2 HSCT.DBR

 ARCHIVE

.  Set General Parameters

.  ======================

   SET LDI          = 1
   SET LDI/ARCHIVE  = 2
   SET DATA         = ES7P_SHELL.HC14_B0000000000
   SET FRAME        = GLOBAL
   SET COMPONENT    = 3
   SET BEG_MESH     = 0
   SET END_MESH     = 4

   SET TAG          = 442

   NODES
      NODE 442
      ARCHIVE

   VECTOR
      NVT_NAME = NODAL.DISPLACEMENT
      ARCHIVE
```

The next example illustrates how to search the displacement vector for the maximum component. This operation requires no input node; instead the result of the operation will report which node and component contain the maximum or minimum value. The result is written to the computational database since the resultant database has the same logical device index as the computational database.

```
*open 1 HSCT.DBC

 ARCHIVE
   SET LDI          = 1
   SET LDI_R        = 1
   SET DATA         = ES7P_SHELL.HC14_B0000000000
   SET FRAME        = GLOBAL
   SET COMPONENT    = ALL

   SET MESH = 1
   VECTOR
      NVT_NAME = NODAL.DISPLACEMENT
      MAX
      ABS
      OUTPUT_DISP = DISPMAX
      ARCHIVE
   STOP
```

14.4.3 SET Command

14.4.3.1 General Description

The SET commands are used to change input parameters from their default values. Some SET commands have no default value and must always be set. For each command documented in PST there is a list of relevant SET commands.

14.4.3.2 Command Syntax

The general form of the command is as follows:

SET *parameter* = *value*

Table 14.4-30 summarizes all PST set commands, described separately in subsequent subsections.

Table 14.4-30 Summary of SET Commands

Keyword	Default Value	Function
CASENAME	-	
CHILDREN	<false>	Flag indicating whether child elements should be included in an element list.
COMPONENT	0	
COMPRESS	<false>	
CONSTRAINT_SET	1	Constraint Set
DATASET_NAME	-	
DEGEN	<false>	
ECHO	<false>	
ELEMENT_NAME	-	
FILE_NAME	-	
FRAME	Global	
GROUP	0	
INPUT_UNIT	5	
LDI	1	Logical Device Index
LOADSET	1	
LOCATION	INTEG_PTS	
MESH	0	
NVTNAME		
NEN	9	
OBJECT		

Table 14.4-30 Summary of SET Commands (Continued)

Keyword	Default Value	Function
ORIENTATION	0	
OUTPUT_UNIT	6	
STEP	0	
TAG	-	
TITLE	-	
TOLERANCE	0.001	

14.4.3.3 SET CASE_NAME Command

Name used to construct the name of the data libraries for an analysis. The case name is used in conjunction with suffixes .DBC, .DBE, .DBS, and .DBR to create the name of the data libraries.

SET CASE_NAME = *casename*

where

Parameter	Default Value	Description
casename		Name of the case used to construct the data library names. (Ex. HSCT.DBC)

14.4.3.4 SET CONSTRAINT_SET Command

Constraint set number associated with element and nodal data in the reference mesh and all refined meshes.

SET CONSTRAINT_SET = *con_set*

where

Parameter	Default Value	Description
con_set	1	Constraint set number.

14.4.3.5 SET COMPONENT Command

This command is used to specify which nodal components to consider when post-processing the results of an analysis.

SET COMPONENT = *comp*

where

Parameter	Default Value	Description
comp	1,2,3,4,5,6	Nodal component.

14.4.3.6 SET COMPRESS Command

This command sets a flag which compresses the node and element numbering such that no gaps exist. Only active nodes and elements are considered.

SET COMPRESS = *compress*

where

Parameter	Default Value	Description
compress	<false>	Flag indicating if node and element numbering should be compressed.

14.4.3.7 SET DEGEN Command

When this flag is set to true all triangles are converted to degenerate quads.

SET DEGEN = *degen*

where

Parameter	Default Value	Description
degen	<false>	Converts all triangles to degenerate quads.

14.4.3.8 SET DATASET_NAME Command

SET DATASET_NAME = [*ldi*], { *dsname* | *idsn* }

where

Parameter	Default Value	Description
ldi	1	Logical device index.
dsname		Dataset name.
idsn		Dataset sequence number.

14.4.3.9 SET ECHO Command

This flag controls the display of commands issued to the PST processor. When <true> all commands which change values of parameters are echoed. The echo includes the name of the command and the value of the command.

SET ECHO = *echo*

where

Parameter	Default Value	Description
echo	<false>	Controls the display of PST commands.

14.4.3.10 SET ELEMENT_NAME Command

This command sets the name of the element, which is constructed by concatenating the element processor name with the element type.

SET ELEMENT_NAME = *element_name*

where

Parameter	Default Value	Description
element_name		Name of element.

14.4.3.11 SET FILE_NAME Command

This command is used to set the name of input file or log file needed by some PST commands:

SET FILE_NAME [*/qualifier*] = *filename*

where

Parameter	Default Value	Description
filename		Name of input file or log file.

Table 14.4-31 Description of Qualifiers for SET FILE_NAME Command

/qualifier	Default Value	Description
/DISP	DISP.DAT	Name of PATRAN nodal results file containing nodal displacements.
/FORCE	FORCE.DAT	Name of PATRAN nodal results file containing nodal forces.

Table 14.4-31 Description of Qualifiers for SET FILE_NAME Command (Continued)

/qualifier	Default Value	Description
/FROM		Used to designate the name of the source file when the output of an operation is another file.
/NEUTRAL	MODE.DAT	Name of PATRAN neutral file.
/PRESSURE	PRESSURE.DAT	Name of PATRAN element results file which contains element pressures.
/RESULT	RESULT.DAT	Name of PATRAN results file containing strains, stresses, and strain energy densities.
/TO		Used to designate the name of the destination file when the output of an operation is another file.

14.4.3.12 SET FRAME Command

This command allows you specify the coordinate frame in which an operation should be carried out or in which coordinate system results should be output. Typically the command refers to nodal displacements and forces which are typically stored in the computational frame, but are often displayed with respect to the global reference frame.

SET FRAME = *frame*

where

Parameter	Default Value	Description
frame	Global	Coordinate frame

Valid coordinate frames are displayed in Table 14.4-32.

Table 14.4-32 Valid Options for the SET FRAME Command

Reference Frame	Description
Global	Indicates that the operation to be completed is conducted in the global cartesian coordinate system (x_g, y_g, z_g).
Computational	Indicates that the operation to be completed is conducted in the computational frame of each node (x_c, y_c, z_c).

14.4.3.13 SET GROUP Command

This command allows you to select which element groups to include for a particular operation:

SET GROUP = *group*

where

Parameter	Default Value	Description
group	0	Element Group (0 implies process all groups)

14.4.3.14 SET INPUT_UNIT Command

Fortran input unit used by processor PST.

SET INPUT_UNIT = *input*

where

Parameter	Default Value	Description
input	5	FORTRAN input unit

14.4.3.15 SET LDI Command

This command sets the logical device index of the data library to be used by subsequent operations. It is assumed that the data library has been opened with the *OPEN directive before it is accessed by PST.

SET LDI[*/qualifier*] = *ldi*

where

Parameter	Default Value	Description
ldi	1	Logical device index

The qualifier is optional and is primarily used to designate alternate data library destinations for output generated. A summary of valid qualifiers is shown in Table 14.4-33.

Table 14.4-33 Description of Qualifiers for SET LDI Command

/qualifier	Description
/ARCHIVE	Logical device index of archival data library.
/FROM	Used to designate the source data library when the input of an operation is another data library.
/TO	Used to designate the destination data library when the output of an operation is another data library.

14.4.3.16 SET LOAD_SET Command

This command changes the default load set number to be used in constructing dataset names within the ARCHIVE sub-processor. In the TRANSLATE processor this command is used to select load set from other FEM analyzers.

SET LOAD_SET = *load_set*

where

Parameter	Default Value	Description
load_set	1	Load Set

14.4.3.17 SET LOCATION Command

SET LOCATION = *location*

where

Parameter	Default Value	Description
location	INTEG_PTS	

14.4.3.18 SET MESH Command

This command is used to designate which mesh to archive and extract data for.

SET MESH = {*beg_mesh, end _mesh, inc_mesh* | *ALL* }

where

Parameter	Default Value	Description
beg_mesh	0	Beginning mesh number
end_mesh	0	Ending mesh number
inc_mesh	1	Mesh increment between beginning and ending mesh number

If only the *beg_mesh* parameter is specified, *end_mesh* = *beg_mesh*. If *inc_mesh* is omitted it is always assumed to be one. Alternatively you can set SET MESH = ALL which will automatically includes all the mesh contained in the data library in subsequent operations.

14.4.3.19 SET NVT_NAME Command

Name of nodal vector. The vector has to be in NVT format.

SET NVT_NAME = *nvt_name*

where

Parameter	Default Value	Description
nvt_name		Name of nodal vector.

14.4.3.20 SET OBJECT Command

SET OBJECT = [*ldi*], { *dsname* | *idsn* }

where

Parameter	Default Value	Description
object		

14.4.3.21 SET ORIENTATION Command

SET ORIENTATION = *orient*

where

Parameter	Default Value	Description
orient	0	Fabrication orientation

14.4.3.22 SET OUTPUT_UNIT Command

SET OUTPUT_UNIT = *output*

where

Parameter	Default Value	Description
output	6	FORTRAN output unit.

14.4.3.23 SET STEP Command

Set step number. For linear static solutions the step number is zero.

SET STEP = *step*

where

Parameter	Default Value	Description
step	1	Step number

14.4.3.24 SET TAG Command

Set tag character string to uniquely identify record groups. String is limited to 6 characters.

SET TAG = *tag*

where

Parameter	Default Value	Description
tag	None	Character string used as tag for record names.

14.4.3.25 SET TITLE Command

Set title of analysis. The title should be enclosed in quotes.

SET TITLE = *"title"*

where

Parameter	Default Value	Description
"title"	None	Title given to analysis.

14.4.3.26 SET TOLERANCE Command

Tolerance to be used when searching for nodes in a user-specified coordinate range.

SET TOLERANCE = *tolerance*

where

Parameter	Default Value	Description
tolerance	1.0e-03	

14.4.3.27 Input Datasets

Some SET commands access HDB data objects.

Table 14.4-34 SET Command Input Datasets

Dataset	Type	Description
CSM.SUMMARY...*mesh*	CSM	Complete Summary of the Model (CSM data object)

14.4.3.28 Output Datasets

None.

14.4.3.29 Limitations

None.

14.4.3.30 Error Messages

The error messages for the SET command are informative and self-explanatory.

14.4.3.31 Examples and Usage Guidelines

See Examples and Usage Guidelines of other commands for illustrations on SET command usage.

14.4.4 STOP Command

The STOP command terminates the processor. Any open HDB or DB data objects will be closed gracefully and main memory blocks allocated through MEM will be released to the system. If PST was called from a macro-processor then control is returned to it, otherwise control is returned to UNIX. The COMET-AR command "RUN *processor_name*" will perform this same function (in addition to transferring control to the new processor).

```
STOP
```

14.4.4.1 Input Datasets

None.

14.4.4.2 Output Datasets

None.

14.4.4.3 Limitations

None.

14.4.4.4 Error Messages

None.

14.4.4.5 Examples and Usage Guidelines

The following is an example of how the STOP command is used to terminate a processor.

```
*open 1 HSCT.DBC

SET NEUT = hsct.pat

 TRANSLATE
   FROM COMET-AR TO PATRAN
   MODEL
   TRANSLATE
 STOP
```

14.4.5 TRANSLATE Sub-Processor

14.4.5.1 General Description

The TRANSLATE sub-processor (previously called CONVERT) allows the user to translate data between different FEM analyzers and external modeling tools. Currently the translator recognizes COMET-AR, PATRAN, and STAGS 2.0 (QSTAGS) data formats. The translator converts data from the source data format to a temporary intermediate data format, and then to the destination data format. By adopting an intermediate data format, each additional data format added to the PST immediately has access to all other data formats.

14.4.5.2 Command Syntax

The general format of a translation command stream is shown below.

```
TRANSLATE
      FROM    from_data_format
              .
      from_options
              .
      TO      to_data_format
              .
      to_options
              .
      TRANSLATE
      RETURN
```

The TRANSLATE sub-processor is invoked by issuing the TRANSLATE command at the main command level. Once you have entered the TRANSLATE sub-processor you have to specify which data formats to translate "from" and "to" via the FROM and TO commands, augmented by keywords referred to as *from_options* and *to_options* used to select finite element quantities to translate. A brief description of each command type shown above is shown in Table 14.4-35.

Table 14.4-35 Summary of Primary TRANSLATE Commands

Command Name	Function
FROM from_data_format	The FROM command is used to specify the source data format. Currently supported data formats are listed in Table 14.4-36.
from_options	The *from_options* indicate which finite element quantities to translate. For each finite element quantity in the database there is a keyword which controls whether this quantity should be translated, usually set to <true> or <false>. The keywords are listed in Table 14.4-38.
RETURN	This command returns program control to the main command level.
TO to_data_format	The TO command is used to specify the destination data format. Currently supported data formats are listed in Table 14.4-37.
to_options	See *from_options*.
TRANSLATE	This command initiates the actual translation process.

PST can read the data formats shown in Table 14.4-36.

Table 14.4-36 Summary of FROM Data Formats

Data Format	Description
COMET-AR	This FROM data format reads the contents of a COMET-AR database. The translator is capable of reading all HDB data objects generated by a standard model definition procedure. It can also read output contained in NVT and EST data objects.

Table 14.4-36 Summary of FROM Data Formats (Continued)

Data Format	Description
PATRAN	This FROM data format reads the content of the PATRAN neutral file. The translator can read PATRAN Packet Types: 01, 02, 05–08 and 14.
STAGS	This FROM data format reads the content of the STAGS database directly via STAR access routines.

PST can write to the data formats shown in Table 14.4-37.

Table 14.4-37 Summary of TO Data Formats

Data Format	Description
CLIP	This TO data format generates a set of CLAMP *ADD files which are read by a generic model procedure. Not all finite element quantities can be generated using this data format.
COMET-AR	This TO data format is capable of generating all finite element quantities required to define a model in COMET-AR. The model generated by translation can subsequently be used to perform adaptive mesh refinement.
PATRAN	This TO data format generates a PATRAN neutral file.
STAGS	Not implemented yet.

Once the source and destination data formats have been determined, the user has to indicate which finite element quantities to translate. Table 14.4-38 lists the keywords used to control the translation of model data. The MODEL keyword automatically translates all model quantities from the database. Alternatively you can control the translation of each data type by selectively turning data types on or off.

Table 14.4-38 Summary of Keywords for Controlling Basic Model Finite Element Quantities

Data Type	Description
MODEL	This keyword can be used to designate that all solution data quantities should be translated. When this keyword is used all other keywords associated with solution quantities are automatically set to <true>.
BC	Controls the translation of boundary conditions.
ELEMENT	Controls the translation of element attributes and element connectivities.
ELT_LOAD	Controls the translation of body, line, pressure, and surface loads. Some or all of these data formats may be supported by the translator.
MPC	Controls the translation of multi-point constraints. Not all data formats support MPCs.
NODAL_DISP	Controls the translation of nodal specified displacement vectors. For some data formats all boundary conditions are described via a nodal displacement vector.
NODAL_FORCE	Controls the translation of nodal force vectors. For some data formats all boundary conditions are described via a nodal force vector.

Table 14.4-38 Summary of Keywords for Controlling Basic Model Finite Element Quantities

Data Type	Description
NODE	Controls the translation of nodes. The node number, coordinate location, and coordinate system in which the node is located is translated.
TRIAD	Controls the translation of nodal transformations.

Table 14.4-39 summarizes all the keywords used to control the translation of solution quantities, which can be translated independent of model quantities. The SOLUTION keyword automatically translates all solution quantities from the database. Alternatively the user can control the translation of each solution data type by selectively turning data types on or off.

Table 14.4-39 Summary of Keywords for Controlling Basic Solution Finite Element Quantities

Data Type	Description
SOLUTION	This keyword can be used to designate that all solution quantities should be translated. When used all other keywords associated with solution quantities are automatically set to <true>.
DISPLACEMENT	Controls the translation of displacement quantities.
ENERGY	Controls translation of strain energy densities. The location at which strain energy density is extracted or generated depends on the TO and FROM data formats used in translation process.
ERROR	Controls the translation of element errors.
STRAIN	Controls the translation of the strain resultants. The location at which the strains are extracted or generated depends on the TO and FROM data formats used in the translation process.
STRESS	Controls the translation of the stress resultants. The location at which the stresses are extracted or generated depends on the TO and FROM data formats used in the translation process.

The restrictions on data translation between different data formats is shown in Table 14.4-40. This table consists of two main columns: a TO column and a FROM column. Under each column there are additional columns for each data format. For each data format the data types that PST can translate are checked off.

Table 14.4-40 Summary of Translation Capabilities

	FROM			TO			
	COMET-AR	PATRAN	STAGS	CLIP	COMET-AR	STAGS	PATRAN
	Model						
Node	✔	✔	✔	✔	✔		✔
Triad	✔	✔	✔		✔		✔
Element	✔	✔	✔	✔	✔		✔
Element Load	✔	✔	✔	✔	✔		✔

Table 14.4-40 Summary of Translation Capabilities (Continued)

	FROM			TO			
	COMET-AR	PATRAN	STAGS	CLIP	COMET-AR	STAGS	PATRAN
Boundary Condition	✔	✔	✔	✔	✔		✔
Multi-Point Constraint	✔						✔
Nodal Force	✔	✔	✔		✔		✔
Nodal Displacement	✔	✔	✔		✔		✔
Material							
Fabrication							
	Solution						
Displacement	✔		✔		✔		✔
Force	✔		✔				
Error	✔						
Strain	✔		✔		✔		✔
Stress	✔		✔		✔		✔
Energy	✔		✔		✔		✔
Pressure							✔
	MISC						
Element Check							
Interpolation/Extrapolation					✔		

14.4.5.3 The FROM COMET-AR Command

14.4.5.3.1 General Description

This command sets the source data format to COMET-AR. A COMET-AR data library containing model and solution data is subsequently read when the TRANSLATION command is issued. It is assumed that the data library has previously been opened with the *OPEN directive.

14.4.5.3.2 Command Syntax

```
FROM COMET-AR
```

14.4.5.3.3 Relevant SET Commands

Table 14.4-41 SET Commands Relevant to the FROM COMET-AR Command

Command Name	Default	Function
SET CONSTRAINT_SET	1	Constraint set to use when translating model and solution data.
SET LDI	1	Logical device index of central data library (.DBC) which contains model and solution data.
SET LOAD_SET	1	Load set to use when translating model and solution data.
SET LOCATION	INTEG_PTS	Location of element stress, strain, and strain energy densities used in the translation process.
SET MESH	0	Mesh number from which model and solution data will be read.
SET STEP	0	Solution step from which data will be read.

14.4.5.4 The RETURN Command

14.4.5.4.4 General Description

This command returns program control to the executive command level.

14.4.5.4.5 Command Syntax

RETURN

14.4.5.5 The TO PATRAN Command

14.4.5.5.6 General Description

This command sets the destination data format to PATRAN. A Patran neutral file and results file is generated when the TRANSLATION command is issued. The name of the Patran neutral file and results file is specified with SET commands.

14.4.5.5.7 Command Syntax

TO PATRAN

14.4.5.5.8 Relevant SET Commands

Table 14.4-42 SET Commands Relevant to the TO PATRAN Command

Command Name	Default	Function
SET COMPRESS	<false>	Compresses node and element numbering to prevent gaps in numbering.
SET FILE/NEURTAL	MODEL.DAT	Sets name of Patran neutral file.
SET FILE/RESULTS	STRESS.DAT	Sets name of Patran results file containing stress, strain, and strain energy resultants.
SET LOAD_SET	1	Load set to use when translating model and solution data.
SET TITLE	" "	Title given to model; should be enclosed in quotes.

14.4.5.6 The TRANSLATE Command

14.4.5.6.1 General Description

This command, issued from the TRANSLATE sub-processor, initiates the translation of data from the source data format to the destination data format.

14.4.5.6.2 Command Syntax

TRANSLATE

14.4.5.7 Examples and Usage Guidelines

Examples of different translation scripts can be found in the $AR_PST/example directory.

14.4.6 References

[1] Felippa, C., *A Command Language for Applied Mechanics Processors, Volume I: The Language*, NASA CR-178384, 1988.

[2] Felippa, C., *A Command Language for Applied Mechanics Processors, Volume II: Directives*, NASA CR-178385, 1989.

[3] Stanley, G. and Swenson, L., *HDB: Object-Oriented Database Utilities for COMET-AR*, NASA Computational Structural Mechanics (CSM) Contract Report, August 1992.

Part IV

DATABASE

In this part of the COMET-AR User's Manual, we describe the database from an engineering analysis user's perspective. The structure and function of database files and datasets (or data objects) created by COMET-AR is presented in enough detail that you can monitor the progress of the analysis by examining the database.

15 Database Summary

15.1 Overview

In this chapter, those aspects of the COMET-AR database related to data descriptions, access methods, and organization are described (see Table 15.1-1). In Section 15.2, a summary is presented of data objects (i.e., data structures and corresponding data management utilities) that are used in COMET-AR analysis, with particular emphasis on those data objects used for adaptive refinement applications. The parameters maintained in the complete summary of the model table (CSM), the element and line refinement tables (ERT and LRT), the element geometry table (EGT), and the element error table (EET) are described. These data objects are employed primarily by the error estimation (ERRi) and mesh refinement (REFi) processors. Discussion in Section 15.3 addresses how these and other data objects may be accessed, from both the procedure and processor levels. In Section 15.4, a description of how the database evolves during adaptive refinement is presented.

Table 15.1-1 Outline of Chapter 15: Database Summary

Section	Title
15.2	Data Objects
15.3	Database Access
15.4	Database Organization and Evolution

15.2 Data Objects

In this section we review the concept of data objects, explain how the COMET-AR database has been mapped onto this conceptual framework via the high-level database (HDB) utilities [1], and briefly describe those data objects which are central to COMET-AR usage.

A data object is defined as the combination of a data structure and its associated data-specific manipulation utilities. By data structure we mean the organization of the data, which has both logical (i.e., conceptual) and physical (i.e., actual) characteristics. By data utilities, we mean FORTRAN subroutines that are specifically designed for particular data structures. Data class denotes a generic description of a particular type of data object; that is, a data object is a specific instance of a data class. For example, many data objects of the same class may be in the database at any given time.

Logically, an HDB data object is typically represented as a rectangular table, with object-specific attributes (nodal coordinates, element nodes, etc.) recorded along the rows, and an index label (node number, element number, etc.) along the columns. Physically, a data object involves the mapping of the logical table to a set of Global Access Library (GAL) records and record groups [2]. This mapping is performed automatically by the high-level (object-oriented) HDB utilities, so that HDB users do not have to be concerned with such transformation details. HDB utilities and physical data structures are discussed in detail in Reference [1].

High-level (HDB) object-based utilities are provided for each object class. These utilities allow you to imagine each object in its logical data format, e.g., as a simple table structure. In turn, these HDB utilities invoke the intermediate-level database (DB) utilities [3], which create and access GAL physical data structures and perform automatic local memory buffering functions. A conceptual view of a data object is shown in Figure 15.2-1.

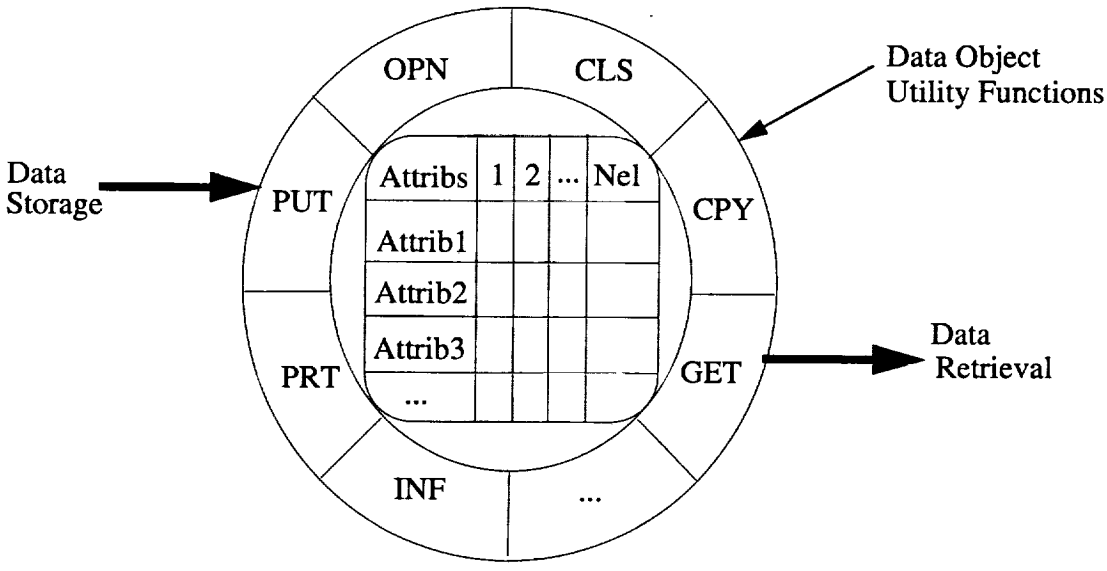


Figure 15.2-1 Conceptual View of a Data Object

Figure 15.2-1 shows a generic data structure surrounded by a number of support utilities such as OPN (open data object), CLS (close data object), GET (get data from data object), PUT (put data into data object), INF (retrieve information about data object), PRT (print data object), and CPY (copy data object). Additional tailor-made support utilities may be present for some data objects as necessary. The implementation of this concept provides both the COMET-AR user and software developer with all of the tools needed to manipulate the database.

15.2.1 Summary of Primary COMET-AR Data Objects

The implementation of data objects in COMET-AR is performed by organizing the entire database into data objects and developing the necessary support utilities. The high-level data object support utilities illustrated in Figure 15.2-1 are referred to collectively as HDB and are described in detail in Reference [1]. Beneath the HDB level are a set of generic global/local data management utilities referred to as DB. The DB utilities are described in detail in Reference [3]. Below the DB level is the COMET-AR global data manager, GAL. The GAL description of the database in terms of data libraries, data sets, and data records is preserved throughout the DB and HDB levels, except that the HDB level hides the details of record organization and makes the data object appear to the user as a simple, logical table regardless of the underlying physical (GAL) structure.

The correspondence between data objects and datasets is one-to-one. Every data object corresponds to a dataset and vice versa. The COMET-AR database for a given problem, which may be distributed over one or more data libraries, is comprised of data objects belonging to a class that falls into one of the following general categories:

- Summary Data Objects;
- Nodal Data Objects;
- Element Data Objects;
- Line Data Objects;
- Surface Data Objects; or
- System Data Objects.

A summary of specific data object dataset names for each of these class categories is given in Tables 15.2-1 through 15.2-6. Specific data objects may or may not exist in the database for a particular analysis. Each data object is named according to its class category (Summary, Element, Nodal, Line, etc.), a class component name (Definition, Errors, Geometry, Coordinates, etc.), and one or more qualifiers for object uniqueness (analysis step, load set, constraint set, or mesh number). In the following naming conventions, uppercase words are required, italicized capitalized words are user dependent, and lowercase italicized words (numerical qualifiers) are typically analysis dependent.

Table 15.2-1 Summary Data Object Dataset Names

Class	Description	Generic Dataset Name	Example Dataset Name
ANS	Analysis Summary	ANALYSIS.SUMMARY	ANALYSIS.SUMMARY
ARS	Adaptive Refinement Summary	REFINEMENT.SUMMARY	REFINEMENT.SUMMARY
CSM	Complete Model Summary	CSM.SUMMARY...*mesh*	CSM.SUMMARY...2

Table 15.2-2 Element Data Object Dataset Names

Class	Description	Generic Dataset Name	Example Dataset Name
EAT	Element Attributes (Generic)	*EltName.AttName.step..mesh* *EltName.AttName.ldset.cnset.mesh*	ES1_EX47.COUP_STIFF.1..2 ES1_EX47.COUP_STIFF.2.2.1
EDT	Element Definition	*EltName*.DEFINITION...*mesh*	ES7P_SHELL.DEFINITION...3
EET	Element Error	*EltName*.ERROR.*step..mesh* *EltName*.ERROR.*ldset.cnset.mesh*	ES36_MIN3.ERROR.2..1 ES36_MIN3.ERROR.1.2.3
EFT	Element Fabrication	*EltName*.FABRICATION...*mesh*	ES1_EX97.FABRICATION
EGT	Element Geometry	*EltName*.GEOMETRY...*mesh*	ES1_EX96.GEOMETRY...4
EIT	Element Interpolation	*EltName*.INTERPOLATION...*mesh*	ES7P_SHELL.INTERPOLATION...1
ELT	Element Loads	*EltName*.LOAD.*step..mesh* *EltName*.LOAD.*ldset..mesh*	ES1_EX42.LOAD.3..5 ES1_EX42.LOAD.1..4
EMT	Element Matrix	*EltName.MatName...mesh*	ES1_EX47.STIFFNESS...3
ERT	Element Refinement	*EltName*.REFINEMENT...*mesh*	ES36_MIN3.REFINEMENT...2
EST	Element Stress/Strain	*EltName.StrName.step..mesh* *EltName.StrName.ldset.cnset.mesh*	ES1P_SHELL.STRAIN.1.1 ES1P_SHELL.STRESS.2.1.4

Table 15.2-3 Nodal Data Object Dataset Names

Class	Description	Generic Dataset Name	Example Dataset Name
NAT	Nodal Attributes (Generic)	NODAL.*AttName.step..mesh* NODAL.*AttName.ldset.cnset.mesh*	NODAL.NORMAL.1..2 NODAL.NORMAL.2.2.1
NCT	Nodal Coordinates	NODAL.COORDINATE...*mesh*	NODAL.COORDINATE...2
NDT	Nodal DOF	NODAL.DOF..*cnset.mesh*	NODAL.DOF..2.3
NOT	Nodal Ordering	NODAL.ORDER...*mesh*	NODAL.ORDER...1
NTT	Nodal Transformation	NODAL.TRANSFORMATION...*mesh*	NODAL.TRANSFORMATION...2
NVT	Nodal Vector	NODAL.*VecName.step..mesh* NODAL.*VecName.ldset.cnset.mesh*	NODAL.DISPLACEMENT NODAL.VELOCITY.2.1.3

Table 15.2-4 Line Data Object Dataset Names

Class	Description	Generic Dataset Name	Example Dataset Name
LRT	Line Refinement Table	LINE.REFINEMENT...*mesh*	LINE.REFINEMENT...2

Table 15.2-5 Surface Data Object Dataset Names

Class	Description	Generic Dataset Name	Example Dataset Name
SRT	Surface Refinement Table	SURFACE.REFINEMENT...*mesh*	SURFACE.REFINEMENT...3

Table 15.2-6 Matrix/Vector Data Object Dataset Names

Class	Description	Generic Dataset Name	Example Dataset Name
SMT	System Matrix Table	SYSTEM.*MatName...mesh*	SYSTEM.STIFFNESS...4
SVT	System Vector Table	SYSTEM.*VecName...mesh*	SYSTEM.VELOCITY...3

In the tables above, the Class column denotes the abbreviated names used to describe the data object class name. For example, an Element Definition Table is a data object in class EDT.

15.2.2 General Description of Data Objects

Data objects maintained within a COMET-AR database can logically be viewed as consisting of tables with class-specific attributes labelling rows and each column comprising a record of information indexed by a value meaningful for the class. For example, nodal-class objects are column-indexed by node numbers, while element-class objects are column-indexed by element numbers. Each class of data objects is generally described in terms of a tabular organization listing its attributes names (row labels) and column-indexing parameters (column labels) as shown below in Table 15.2-7.

Table 15.2-7 Generic Data Object Description

Attribute	Column_1	...	Column_N
Name1	Name1_1	...	Name1_N
Name2	Name2_1	...	Name2_N
...	...	...	...
NameM	NameM_1	...	NameM_N

In the following subsections, representative COMET-AR data object descriptions are presented. Each data object is described in two tables. Table 15.2-8 lists the data object's attribute names (in the tabular form described above) and Table 15.2-9 lists the attribute's characteristics (data types and dimensions) and meaning. The data objects listed in Table 15.2-8 are central to COMET-AR usage and are described in more detail beginning in subsection 15.2.2.2.

Table 15.2-8 Data Objects Selected For Presentation

Class	Description	Generic Dataset Name
CSM	Complete Summary of the Model	MODEL.SUMMARY..*mesh*
EDT	Element Definition Table	*EltName*.DEFINITION...*mesh*
EGT	Element Geometry Table	*EltName*.GEOMETRY...*mesh*
ELT	Element Loads Table	*EltName*.LOADS.*step*..*mesh*
ERT	Element Refinement Table	*EltName*.REFINEMENT...*mesh*
LRT	Line Refinement Table	*EltName*.REFINEMENT...*mesh*
NDT	Nodal DOF Table	NODAL.DOF..*cnset.mesh*

Before discussing the logical structure of data objects in terms of their tabular structure and attribute names, the next subsection presents the physical storage of a data object as a collection of GAL records.

15.2.2.1 Storage of Data Objects in a GAL Dataset

A dataset containing a single HDB data object consists of five records and one record group. The five records contain descriptive information about the object, and the record group contains the data object's values. These records and their characteristics are outlined in Table 15.2-9.

Table 15.2-9 Physical Structure for a Generic Data Object

Record Name	Type	Length	Description
CONTENTS	Char	20	Class Name: *Class* (e.g., CSM, EDT, etc.)
*Class*_ATTS	Int	5*NATTS	Attribute Descriptor
*Class*_NATTS	Int	1	Number of Attributes (NATTS)
*Class*_PARS	Int	NPARS	Parameters Record
*Class*_NPARS	Int	1	Number of Parameters (NPARS)
Class.1:Ncol	Float/Int	*RecLen*	Data Object Values

All data objects are stored using the same representation scheme. A single object is described by a contents record, an attribute descriptor, a parameter record, and a single record group containing all of the data object's values.

A CONTENTS record contains the class name of the data object in the dataset. The value of a CONTENTS record is text, generically called *Class*, where *Class* represents any of the object classes: CSM, EDT, NDT, etc.

Each data object has an attribute descriptor, called *Class*_ATTS, that contains information about the object's attributes (see Table 15.2-10). Each attribute is described by five numerical parameters: record index, dimension, maximum dimension, symbolic name, and data type. The

*Class*_ATTS record contains five numbers for each attribute, and the number of attributes is stored in the record *Class*_NATTS.

Table 15.2-10 Attribute Parameters for a Generic Data Object

Parameter	Attribute 1	...	Attribute NATTS
Index	$Index_1$	...	$Index_{NATTS}$
Dimension	$Dimension_1$	...	$Dimension_{NATTS}$
Max-Dimension	$Max\text{-}Dimension_1$	...	$Max\text{-}Dimension_{NATTS}$
Symbolic-Name	$Symbolic\text{-}Name_1$	...	$Symbolic\text{-}Name_{NATTS}$
Data-Type	$Data\text{-}Type_1$	...	$Data\text{-}Type_{NATTS}$

The value of an *Index* parameter gives an attribute's word location in a data record where the first word is numbered 1. Each attribute has its own data type but data records have a uniform type and all record attributes are converted to that type for storage. Specifying attribute location by record index is independent of the individual attribute data types.

When positive, a *Dimension* parameter indicates an attribute's current dimension. The value of this parameter may vary among data objects belonging to the same class. For example, an attribute may be sized by mesh parameters that change during adaptive refinement (AR) analyses. When an attribute's *Dimension* is positive, data written to the attribute can contain any number of items not exceeding *Max-Dimension*; however, when reading data from an attribute *Dimension* items are always returned.

When negative, a *Dimension* parameter indicates that an attribute's current dimension varies from record to record. For this case, an attribute's dimension is recorded in another attribute (called the dimension-attribute) in the same record as the attribute's data values. The magnitude of a *Dimension* parameter is the index of the dimension-attribute in the *Class*_ATTS record. As an example of this feature, an Element Refinement Table (ERT) has an attribute called *Child* whose variable dimension is recorded in the dimension-attribute called *Nchild*. For each record, the dimension of *Child* is contained in a dimension-attribute called *Nchild*. This feature can be thought of as variable dimensioning. When an attribute has a variable dimension, its dimension is automatically updated with a write operation and used with a read operation. Reading an attribute with a variable dimension always returns the number of data items last written. (No more than *Max-Dimension* data items can ever be written or read.)

A *Max-Dimension* parameter sets a limit on the number of data items that can be written to an attribute. When attribute parameter *Dimension* is positive, data written to an attribute can contain any number of items not exceeding *Max-Dimension*; however, when reading data from an attribute (not variably dimensioned), *Dimension* items are always returned.

A *Symbolic-Name* parameter encodes an attribute's name using a Q-symbol value. Every attribute name for all data objects has an associated Q-symbol [1]. Identification of attributes by *Symbolic-Name* is the only mechanism used across the HDB class utilities interface.

A *Data-Type* parameter encodes an attribute's data type, again using Q-symbol values. Five data types are currently defined: *qChar*, *qDouble*, *qIntegr*, *qSingle*, and *qSymbol*. Character-valued attributes are stored one character per word using the ASCII encoding standard. Symbol attributes are stored as integer numbers (Q-symbol values) but are displayed by the HDB Print Processor by Q-symbol name. (See Section 14.3, *Database Print Utility,* for a description of the object-oriented HDB print processor.)

An HDB data object's parameter record, *Class*_PARS, contains extraneous information such as values of special parameters supplied when an object is created. For example, most HDB element objects have a *Reserve* parameter that is stored in their parameter record. The size of *Class*_PARS is stored in the record *Class*_NPARS.

Values for an HDB data object (belonging to class *Class*) are stored in one record group called *Class*. All attribute values for a logical HDB data object record are stored in this single record group record.

The data type of a *Class* record group is determined by the data object's attribute data types. If one or more of the attributes is typed *qDouble*, then the record group's data type is *qDouble* and all attribute values are converted to *qDouble* for storage. If one or more of the attributes is typed *qSingle*, then the record group's data type is *qSingle* and all attribute values are converted to *qSingle* for storage. Otherwise, all of the attributes are either *qChar*, *qIntegr*, or *qSymbol* and the record group's data type is *qIntegr*. (*qChar* attribute values are always stored one character per word using the ASCII encoding standard.)

A *Class* record is stored in a memory buffer allocated by the DB Memory Manager (MEM) [3]. Access to attribute values in a *Class* record (either reading or writing) may require conversion between the *Class* record's data type and the attribute's data type. This conversion is automatically performed by HDB using information contained in the *Class*_ATTS attribute descriptor record.

Storing logical attribute values in a common physical record has the property of data locality: related data values are stored physically close together. If an attribute's value is accessed in a record, then it is likely that an associated attribute's value in the same record will also be accessed. When record attributes are processed collectively, disk performance using this single-record approach is better than when using a scheme where each data object attribute is stored in a separate record. This latter approach requires multiple input/output operations for each attribute accessed.

15.2.2.2 CSM — Complete Summary of the Model

The CSM class summarizes all high-level information relevant to the current COMET-AR model. A data object in this class is partitioned into two tables: general summary attributes and element type summary attributes (which range over the number of element types). There is typically only one CSM data object for each model which must be opened first before opening related data objects, such as objects from the EDT or NDT classes. Objects from these classes contain data that varies from element-to-element, or node-to-node. This is because parameters contained in a

CSM data object are automatically employed to dimension attributes appearing in these other objects. Model summary parameters such as the total number of nodes (*Nnode*) and the total number of elements in each element type (*Nelt[EltTyp]*) are automatically updated in the associated CSM object whenever any element or nodal data object is expanded. Information in the CSM class should be extended as necessary to give a complete overview of the model and to permit convenient access at the CSM procedure level. To facilitate communication between COMET-AR procedures and the database, an interface procedure *CSMget* has been written to fetch selected CSM attribute values.

The attributes stored in general summary attributes are shown in Table 15.2-11 and those stored in element type summary attributes are shown in Table 15.2-12.

Table 15.2-11 CSM General Summary Table

General Summary Attributes	
AnaTyp	Analysis Type
ARflag	Adaptive Refinement Flag
CnsFrq	Constitutive Stiffness Archival Frequency (Steps)
DisFrq	Displacement Archival Frequency (Steps)
Dofn (Ndofn)	Nodal DOF Types
HisFrq	Constitutive History Archival Frequency (Steps)
Mesh	Current Mesh
Miters	Maximum Nonlinear Iterations per Step
Mmesh	Maximum Number of Meshes
Mrefin	Maximum Number of Refinement Levels
Msize	Maximum Problem Size (MW)
Mstep	Maximum Number of Steps
Mtime	Maximum Execution Time
Ndofn	Number of Nodal DOF Types
Neq	Number of Equations
Net	Number of Element Types
Nline	Number of Element Lines in LRT
Nload	Number of Load Sets
Nmesh	Number of Meshes
Nnode	Number of Nodes
Nstep	Number of Analysis Steps
Nsurf	Number of Element Surfaces in SRT
Pglob	Current Polynomial Order for Global *p*-Refinement
Size	Current Problem Size (MW)
SolMod	Solid Model Type
Step	Current Step
StfFrq	Stiffness Formation Frequency (Steps)

Table 15.2-11 CSM General Summary Table (Continued)

General Summary Attributes	
StnFrq	Strain Archival Frequency (Steps)
StrFrq	Stress Archival Frequency (Steps)
Maximum Dimensions for AR Storage Reservations	
Mdofn	Maximum Number of Nodal DOF Types
MEchld	Maximum Number of Child Elements
MLchld	Maximum Number of Child Lines
Mnline	Maximum Number of Nodes per Line
Mnsurf	Maximum Number of Nodes per Surface
Mparse	Maximum Number of Element Research Parameters
MPglob	Maximum Polynomial Order for Global *p*-refinement
MSchld	Maximum Number of Child Surfaces

Table 15.2-12 CSM Element Type Summary Table

Attribute	Element Type 1	...	Element Type Net
Class	$Class_I$	...	$Class_{Net}$
Consti	$Consti_I$	...	$Consti_{Net}$
Dim	Dim_I	...	Dim_{Net}
Dofe (Ndofe)	Dofe $(Ndofe_1)_1$	...	Dofe $(Ndofe_{Net})_{Net}$
DrlDof	$DrlDof_1$	...	$DrlDof_{Net}$
DrlOpt	$DrlOpt_1$	...	$DrlOpt_{Net}$
DrlStf	$DrlStf_1$	...	$DrlStf_{Net}$
DrlTol	$DrlTol_1$	...	$DrlTol_{Net}$
EltNam	$EltNam_1$	...	$EltNam_{Net}$
EltPro	$EltPro_1$	...	$EltPro_{Net}$
EltTyp	$EltTyp_1$	...	$EltTyp_{Net}$
KinTyp	$KinTyp_1$	...	$KinTyp_{Net}$
Mee	Mee_1	...	Mee_{Net}
Men	Men_1	...	Men_{Net}
Mip	Mip_1	...	Mip_{Net}
Mnlt	$Mnlt_1$	...	$Mnlt_{Net}$
Mnst	$Mnst_1$	...	$Mnst_{Net}$
Mstore	$Mstore_1$	...	$Mstore_{Net}$
Ndofe	$Ndofe_1$	...	$Ndofe_{Net}$
Nee	Nee_1	...	Nee_{Net}
Nelt	$Nelt_1$	...	$Nelt_{Net}$
Nen	Nen_1	...	Nen_{Net}

Table 15.2-12 CSM Element Type Summary Table (Continued)

Attribute	Element Type 1	...	Element Type Net
NfabEc	$NfabEc_1$	...	$NfabEc_{Net}$
NfabOr	$NfabOr_1$	...	$NfabOr_{Net}$
Ngrp	$Ngrp_1$	...	$Ngrp_{Net}$
Nip	Nip_1	...	Nip_{Net}
Nle	Nle_1	...	Nle_{Net}
NLgeom	$NLgeom_1$	...	$NLgeom_{Net}$
NLload	$NLload_1$	...	$NLload_{Net}$
NLmatl	$NLmatl_1$	...	$NLmatl_{Net}$
Nnlt	$Nnlt_1$	...	$Nnlt_{Net}$
Nnst	$Nnst_1$	...	$Nnst_{Net}$
Nparse	$Nparse_1$	...	$Nparse_{Net}$
Nse	Nse_1	...	Nse_{Net}
Nstore	$Nstore_1$	...	$Nstore_{Net}$
Nstr	$Nstr_1$	...	$Nstr_{Net}$
Option	$Option_1$	...	$Option_{Net}$
P	P_1	...	P_{Net}
Parse (Nparse)	Parse $(Nparse_1)_1$	...	Parse $(Nparse_{Net})_{Net}$
Pmax	$Pmax_1$	...	$Pmax_{Net}$
Shape	$Shape_1$	...	$Shape_{Net}$
VarElt	$VarElt_1$	...	$VarElt_{Net}$
VarFab	$VarFab_1$	...	$VarFab_{Net}$

Definitions of the general summary attributes appearing in Table 15.2-11 and of the element type attributes appearing in Table 15.2-12 are provided in Tables 15.2-13 and 15.2-14.

Table 15.2-13 CSM General Summary Attribute Descriptions

Attribute	Length	Type	Description
AnaTyp	1	Int	Analysis type: qNone — no analysis performed yet qLbuck — linear buckling qLdyna — linear dynamics qLstat — linear statics qLvibr — linear vibrations qNLbuck — nonlinear buckling qNLdyna — nonlinear dynamics qNLstat — nonlinear statics qNLvibr — nonlinear vibrations
ARflag	1	Int	Adaptive refinement flag: qOn — adaptive refinement qOff — no adaptive refinement

Table 15.2-13 CSM General Summary Attribute Descriptions (Continued)

Attribute	Length	Type	Description
CnsFrq	1	Int	Constitutive stiffness archival frequency
DisFrq	1	Int	Displacement archival frequency
Dofn	Ndofn	Int	List of active nodal DOF types
HisFrq	1	Int	Constitutive history archival frequency
Mdofn	1	Int	Maximum number of nodal DOF types
MEchld	1	Int	Maximum number of child elements
Mesh	1	Int	Current mesh number
Miters	1	Int	Maximum nonlinear iterations per step
MLchld	1	Int	Maximum number of child lines
Mmesh	1	Int	Maximum number of meshes
Mnline	1	Int	Maximum number of nodes per line
Mnsurf	1	Int	Maximum number of nodes per surface
Mparse	1	Int	Maximum number of element research parameters
MPglob	1	Int	Maximum polynomial order for global *p*-refinement
Mrefin	1	Int	Maximum number of refinement levels
MSchld	1	Int	Maximum number of child surfaces
Msize	1	Int	Maximum problem size (MW)
Mstep	1	Int	Maximum number of steps
Mtime	1	Int	Maximum execution time
Ndofn	1	Int	Number of active nodal DOF types
Neq	1	Int	Number of active equations
Net	1	Int	Number of active element types
Nline	1	Int	Number of element lines in LRT data object
Nload	1	Int	Number of load sets
Nmesh	1	Int	Number of meshes
Nnode	1	Int	Number of nodes
Nstep	1	Int	Number of analysis steps
Nsurf	1	Int	Number of element surfaces in SRT data object
Pglob	1	Int	Current polynomial order for global *p*-refinement
Size	1	Int	Current problem size (MW)
SolMod	1	Int	Solid model type: qUser or qApprox
Step	1	Int	Current step
StfFrq	1	Int	Stiffness formation frequency
StnFrq	1	Int	Strain formation frequency
StrFrq	1	Int	Stress formation frequency

Table 15.2-14 CSM Element Type Summary Attribute Descriptions

Attribute	Length	Type	Description
Class	1	Int	Intrinsic element class: qBeam — Beam (axial force resultants) qShell — Shell (force/moment resultants per unit length qSolid — Solid continuum (pointwise stress components)
Consti	1	Int	Constitutive basis (strain or stress based): qStrain — strains computed by element qStress — stresses computed by element
Dim	1	Int	Number of intrinsic spatial dimensions represented by element nodal connectivity: 1, 2, or 3
Dofe	Ndofe	Int	List of potential DOFs at element nodes, e.g., qD1, qD2, qD3, qTheta1, qTheta2, qTheta3,... (see Nodal DOF Table (NDT) class for all options)
DrlDof	1	Int	Flag indicating the presence of element drilling stiffness: qTrue or qFalse
DrlOpt	1	Int	Flag indicating if artificial drilling stiffness is added: qTrue or qFalse
DrlStf	1	Int	Absolute value of exponent of artificial drilling stiffness scale factor
DrlTol	1	Int	Angle tolerance between computational frames and element normal vector
EltNam	16	Char	Element name; concatenation of EltPro and EltTyp, separated by an underscore, e.g., "ES1_EX97"
EltPro	16	Char	Element processor name, e.g., ES1, ES2, ...
EltTyp	16	Char	Element type name, e.g., EX91, EX97, EX47, ...
KinTyp	1	Int	Kinematic type: q1D or q2D or q3D for Class = qSolid qC0 or qC1 for Class = qShell qC0 or qC1 for Class = qBeam
Mee	1	Int	Maximum number of element equations (for reserve storage)
Men	1	Int	Maximum number of element nodes (for reserve storage)
Mip	1	Int	Maximum number of element integration points (for reserve storage)
Mnlt	1	Int	Maximum total number of line nodes per element (for reserve storage)
Mnst	1	Int	Maximum total number of surface nodes per element (for reserve storage)
Mstore	1	Int	Maximum number of element initial storage entries (for reserve storage)
Ndofe	1	Int	Number of distinct nodal DOF types potentially active within the element; e.g., 3 for a solid continuum element, or 6 for a general shell element
Nee	1	Int	Number of equations per element (0 ⇒ variable number of equations)
Nelt	1	Int	Number of elements (per type)
Nen	1	Int	Number of element nodes (0 ⇒ variable number of nodes)
NfabEc	1	Int	Number of fabrication eccentricities per element point
NfabOr	1	Int	Number of fabrication orientation parameters per element point
Ngrp	1	Int	Number of element groups in current element type
Nip	1	Int	Number of integration points per element (0 ⇒ variable number of integration points)

Table 15.2-14 CSM Element Type Summary Attribute Descriptions (Continued)

Attribute	Length	Type	Description
Nle	1	Int	Number of lines (edges) per element
NLgeom	1	Int	Nonlinear geometry parameter: 0 ⇒ Linear 1 ⇒ Globally nonlinear 2 ⇒ Globally and locally (element) nonlinear
NLload	1	Int	Nonlinear load flag: qFalse or qTrue
NLmatl	1	Int	Nonlinear material flag: initially, qFalse or qTrue; later changed to a GCP internal option (such as qNoHist) if materially nonlinear (i.e., if initially qTrue).
Nnlt	1	Int	Total number of element line nodes (0 ⇒ variable number of line nodes)
Nnst	1	Int	Total number of surface nodes per element (0 ⇒ variable number of surface nodes)
Nparse	1	Int	Number of element research parameters
Nse	1	Int	Number of surfaces per element
Nstore	1	Int	Number of entries required per element for storage in an Element Initialization Table (EIT) data object (0 ⇒ variable number of entries)
Nstr	1	Int	Number of stress/strain components at each element integration point: Class = qBeam — 4 or 6 Class = qShell — 6 or 8 Class = qSolid — 1, 3, 4 or 6
Option	1	Int	Internal element option number
P	1	Int	Current element polynomial order (0 ⇒ variable element order)
Parse	Nparse	Float	Element research parameters
Pmax	1	Int	Maximum element polynomial order for problem
Shape	1	Int	Shape of primary element nodal surface: qLine — beam element qQuad — quadrilateral element face qTri — triangular element face
VarElt	1	Int	Variable element size flag: qFalse — element size same for all elements qTrue — element size varies with element
VarFab	1	Int	Variable element fabrication properties type: qFalse — same for all elements qElt — vary from element to element qIntPt — vary from point to point

15.2.2.3 EDT — Element Definition Table

The Element Definition Table (EDT) has information which is required for adaptive refinement analysis and includes an element nodal DOF pattern array (which may optionally vary from element to element within a given type), which enables either global or local adaptive *p*-refinement. Adaptive *h*-refinement is accommodated by adding more elements to the table and/or rendering old elements inactive via a Status attribute stored in the table (a utility is provided to retrieve the next active element when processing an element loop). Elements in the EDT are never actually deleted, making it possible to retrieve information from previous models generated during adaptive refinement. However, to obtain topological information about current or previous models, including element hierarchical relations, refer to the Element Refinement Table (ERT) class definition.

The logical organization of an Element Definition Table (EDT) data object is shown in Table 15.2-15.

Table 15.2-15 Element Definition Table (EDT)

Attribute	Element 1	...	Element Nelt
DOFtab (Ndofe,Nen)**	DOFtab (Ndofe,Nen_1)$_1$	...	DOFtab (Ndofe,Nen_{Nelt})$_{Nelt}$
GrpElt	$GrpElt_1$	...	$GrpElt_{Nelt}$
GrpNum	$GrpNum_1$	...	$GrpNum_{Nelt}$
Nee*	Nee_1	...	Nee_{Nelt}
Nen*	Nen_1	...	Nen_{Nelt}
Nip*	Nip_1	...	Nip_{Nelt}
Nnl (Nle)**	Nnl (Nle)$_1$	...	Nnl (Nle)$_{Nelt}$
Nnlt*	$Nnlt_1$	...	$Nnlt_{Nelt}$
Nns (Nse)**	Nns (Nse)$_1$	...	Nns (Nse)$_{Nelt}$
Nnst*	$Nnst_1$	...	$Nnst_{Nelt}$
Nodes (Nen)	Nodes (Nen_1)$_1$	...	Nodes (Nen_{Nelt})$_{Nelt}$
Nstore*	$Nstore_1$	...	$Nstore_{Nelt}$
P*	P_1	...	P_{Nelt}
Status	$Status_1$	...	$Status_{Nelt}$
UsrElt	$UsrElt_1$	...	$UsrElt_{Nelt}$

The columns range over the total number of elements of a particular type (*Nelt*), including both active and inactive elements. The attributes marked with an asterisk may either be constant or variable from element to element (depending on the value of *VarElt* in the associated CSM data object). When these values are constant, they may also be obtained from the CSM data object. The attributes marked with a double asterisk are always stored in the EDT, but they too may either be constant or variable. The attribute *VarElt* in the CSM data object determines whether these element attributes (both single and double asterisks) are constant or variable.

The EDT attributes given in Table 15.2-15 are defined in Table 15.2-16.

Table 15.2-16 EDT Attribute Descriptions

Attribute	Length	Type	Description
DOFtab**	(Ndofe,Nen)	Int	DOF pattern at element nodes: 0 ⇒ DOF is inactive 1 ⇒ DOF is active The correspondence of the rows of DOFtab with DOF types is given by the *Dofe* attribute in the CSM data object.
GrpElt	1	Int	Element number relative to present element group
GrpNum	1	Int	Group number of element relative to present element type
Nee*	1	Int	Number of equations per element (≤ *Ndofe*Nen*)
Nen*	1	Int	Number of element nodes
Nip*	1	Int	Number of integration points per element (for stress storage)
Nnl**	Nle	Int	Number of nodes per element line (edge)
Nnlt*	1	Int	Total number of element line nodes (sum of *Nnl*)
Nns**	Nse	Int	Number of nodes per element surface (face)
Nnst*	1	Int	Total number of element surface nodes (sum of *Nns*)
Nodes	Nen	Int	Element node numbers: Node(a) ≤ 0 ⇒ inactive element node
Nstore*	1	Int	Number of entries per element in element initial storage (EIT) data object
P*	1	Int	Element polynomial (p) order
Status	1	Int	Element status flag: qActive ⇒ element is active
UsrElt	1	Int	User element number (typically same as EDT column number)

The EDT attribute dimension parameters appearing in Table 15.2-16 are stored in the CSM data object and defined in Table 15.2-17.

Table 15.2-17 EDT Parameter Descriptions

Parameter	Length	Type	Description
Ndofe	1	Int	Max number of DOFs per element node
Nle	1	Int	Number of lines (edges) per element (≥0)
Nse	1	Int	Number of surfaces (faces) per element (≥0)

15.2.2.4 EGT — Element Geometry Table

An EGT data object is created by a generic element processor (ES*i* processor), and updated/ utilized by the mesh refinement processors (REF*i*). An EGT is typically stored in the database in

datasets named *EltNam*.GEOMETRY...*mesh* (i.e., one for each element type). The purpose of an EGT is to define links between the discrete finite element model and a continuous, or solid-model, description of the structural geometry. If the user selected the "discrete" solid-model interface option (see Chapter 16, *Solid Model Interface*), the EGT becomes irrelevant because the solid-model and discrete-model geometry are then interpreted as one and the same. If the "user-written" SMI option was selected, the EGT then associates each finite element with one or more solid-model volumes, surfaces, and lines.

A summary of the attributes stored in an EGT data object is given in Table 15.2-18. The *LineID* attribute contains a list of solid-model line identifiers for each element line. If the element line does not lie on a solid-model line, then a zero is stored. Similarly, the *SurfID* attribute contains a list of solid model surface identifiers for each element. If the element is a 2-D surface element (e.g., a plate or shell) then there is only one surface ID per element. If the element is a 3-D solid element, then it will have a number of surfaces, some of which may lie on a solid-model surface, and some of which may be interior. For the interior surfaces a zero is stored in *SurfID*. The *SolID* attribute contains a solid-model volume identifier and is relevant only for 3-D elements.

Table 15.2-18 Element Geometry Table (EGT)

Attribute	Element 1	...	Element Nelt
LineID (Nle)	LineID $(Nle)_1$	...	LineID $(Nle)_{Nelt}$
SoliID	$SoliID_1$	...	$SoliID_{Nelt}$
SurfID (Nse)	SurfID $(Nse)_1$	...	SurfID $(Nse)_{Nelt}$

The columns in this table range over the total number of elements of a particular type (*Nelt*), including both active and inactive elements. The attributes (rows) of an EGT and the parameters used to dimension these attributes are described next in Tables 15.2-19 and 15.2-20.

Table 15.2-19 EGT Attribute Descriptions

Attribute	Length	Type	Description
LineID	Nle	Int	Solid model line identifier for each element line 0 ⟹ element line is not associated with a solid-model line
SoliID	1	Int	Solid model volume identifier for each element volume (3D refinement only) 0 ⟹ element volume is not associated with a solid-model volume
SurfID	Nse	Int	Solid model surface identifier for each element surface 0 ⟹ element surface is not associated with a solid-model surface

The EGT attribute dimension parameters are stored in a CSM data object and are defined in Table 15.2-20.

Table 15.2-20 EGT Parameter Descriptions

Parameters	Length	Type	Description

Table 15.2-20 EGT Parameter Descriptions

Nle	1	Int	Number of lines per element
Nse	1	Int	Number of surfaces per element

15.2.2.5 ELT — Element Load Table

A data object belonging to the Element Loads Table (ELT) class potentially contains all element loads for a particular element type and load case. Structural-element loads include distributed line, surface and body forces/moments, as well as temperatures.

The logical organization of an Element Loads Table (ELT) data object is shown in Table 15.2-21.

Table 15.2-21 Element Load Table (ELT)

Attribute	Element 1	...	Element Nelt
Bload (Ndofe,Nen) Bsyst	Bload (Ndofe,Nen)$_1$ Bsyst$_1$	...	Bload (Ndofe,Nen)$_{Nelt}$ Bsyst$_{Nelt}$
Lload (Ndofe,Nnlt) Lsyst	Lload (Ndofe,Nnlt)$_1$ Lsyst$_1$	...	Lload (Ndofe,Nnlt)$_{Nelt}$ Lsyst$_{Nelt}$
Pload (Nnst) Psyst	Pload (Nnst)$_1$ Psyst$_1$	...	Pload (Nnst)$_{Nelt}$ Psyst$_{Nelt}$
Sload (Ndofe,Nnst) Ssyst	Sload (Ndofe,Nnst)$_1$ Ssyst$_1$	...	Sload (Ndofe,Nnst)$_{Nelt}$ Ssyst$_{Nelt}$

The columns range over the total number of elements of a particular type (*Nelt*), including both active and inactive elements. Attributes (rows) of an ELT data object are described in Table 15.2-22.

Not all of the above attributes groups may be present in a particular ELT data object. For example, perhaps only line loads may be defined, or only pressure loads, or the combination of the two. Only those loading types (i.e., attribute groups) relevant to a particular model will be present. This enables element consistent-load generation to be data-driven. It is assumed that "live" (displacement-dependent) loads are defined in a separate dataset from "dead" (displacement-independent) loads.

Table 15.2-22 ELT Attribute Descriptions

Attribute	Length	Type	Description
Bload	(Ndofe,Nen)	Flost	Body loads for a particular element
Bsyst	1	Int	Body load coordinate system: qGlobal, qElemnt, or qNodal
Lload	(Ndofe,Nnlt)	Float	Line loads for a particular element

Table 15.2-22 ELT Attribute Descriptions (Continued)

Attribute	Length	Type	Description
Lsyst	1	Int	Line load coordinate system: qGlobal, qElemnt, or qNodal
Pload	(Nnlt)	Float	Pressure loads for a particular element
Psyst	1	Int	Pressure load coordinate system: qGlobal, qElemnt, or qNodal
Sload	(Ndofe,Nnst)	Float	Surface loads for a particular element
Ssyst	1	Int	Surface load coordinate system: qGlobal, qElemnt, or qNodal

The ELT attribute dimension parameters appearing in Table 15.2-22 are stored in the CSM data object and are defined in Table 15.2-23.

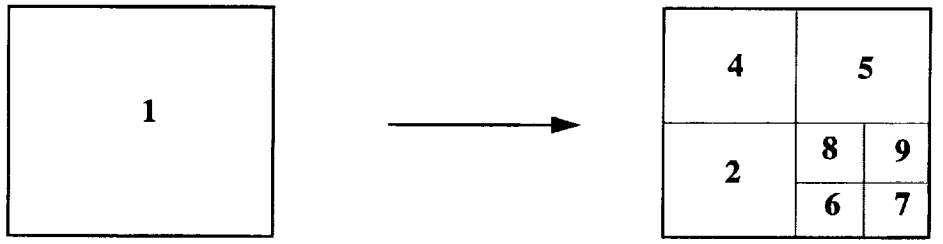
Table 15.2-23 ELT Parameter Descriptions

Parameters	Length	Type	Description
Ndofe	1	Int	Number of (potential) DOFs per element node
Nen	1	Int	Number of element nodes
Nnlt	1	Int	Total number of nodes on all element lines
Nnst	1	Int	Total number of nodes on all element surfaces

15.2.2.6 ERT — Element Refinement Table

A data object belonging to the Element Refinement Table (ERT) class is used to keep track of how elements refine during adaptive *h*- and/or *p*-refinement. An ERT is created and updated by mesh refinement processors such as REFi.

In the case of *h*-refinement, an ERT logically represents a set of tree structures which connect elements that emanate from each element in the initial mesh. When elements subdivide (fission), the new elements thus created are referred to as child elements, and the original element is the parent element. As the process continues, many generations can be created by fission and annihilated by fusion (recombination of a set of child elements back into their parent element). This process is illustrated schematically in Figure 15.2-2 for 2-D quadrilateral elements, which are logically represented as a quad-tree (4 child elements per parent element). The generalization to 3-D brick-type elements is an oct-tree (8 child elements per parent element). During *p*-refinement, an ERT is used just to keep track of the old and new *p* levels for each element. For combined *h*/*p*-refinement, an ERT describes both the *h* nesting and the *p* levels for every element in the mesh (for a given element type).



Element	1	2	3	4	5	6	7	8	9
Parent		1	1	1	1	3	3	3	3
Child1	2		6						
Child2	3		7						
Child3	4		8						
Child4	5		9						

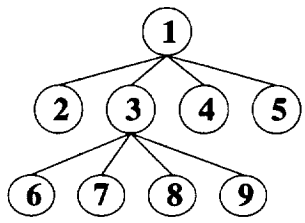


Figure 15.2-2 Quad-tree Growth during Element *h*-Refinement

The main use of this tree structure is for adaptive *h*-refinement, (although an ERT contains data pertinent to *p*- and *h*/*p*-refinement as well). In the case of *h*-refinement, the tree structure is used: (i) to permit interpolation of model and/or solution data between two different meshes; and (ii) to allow the mesh refinement process to be traced backwards in time, as in the case of adaptive unrefinement, where child elements may fuse together and reactivate their parent element. For the solution interpolation function, more than one ERT may be necessary (e.g., to map historical solutions during nonlinear or dynamic analysis from an earlier mesh/step to the current mesh/step). In that case, an ERT would be needed for each pertinent load or time step at which the model had been refined.

The logical organization of an Element Refinement Table (ERT) data object is shown in Table 15.2-24.

Table 15.2-24 Element Refinement Table (ERT)

Attribute	Element 1	...	Element Nelt
Child (Nchild)	Child $(\text{Nchild}_1)_1$	...	Child $(\text{Nchild}_{\text{Nelt}})_{\text{Nelt}}$
Hnew	Hnew_1	...	$\text{Hnew}_{\text{Nelt}}$
Hold	Hold_1	...	$\text{Hold}_{\text{Nelt}}$
Nchild	Nchild_1	...	$\text{Nchild}_{\text{Nelt}}$
Nunref	NunRef_1	...	$\text{NunRef}_{\text{Nelt}}$
Parent	Parent_1	...	$\text{Paren}_{\text{tNelt}}$
Pnew	Pnew_1	...	$\text{Pnew}_{\text{Nelt}}$
Pold	Pold_1	...	$\text{Pold}_{\text{Nelt}}$
pRef	pRef_1	...	$\text{pRef}_{\text{Nelt}}$
Status	Status_1	...	$\text{Status}_{\text{Nelt}}$
UniRef	UniRef_1	...	$\text{UniRef}_{\text{Nelt}}$

Table columns range over the total number of elements of a particular type (*Nelt*), including both active and inactive elements. Attribute fields of an ERT are described below in Table 15.2-25. The dimension of *Child* (the value of attribute *Nchild*) may vary from element to element. The maximum value of *Nchild* is stored as an attribute in the CSM data object, called *MEchld*. Element edge refinement may be indicated even if *Pnew* = *Pold*.

Table 15.2-25 ERT Attribute Descriptions

Attribute	Length	Type	Description
Child	Nchild	Int	Element indices of child elements
Hnew	1	Int	Number of levels of *h*-refinement after current refinement stage
Hold	1	Int	Number of levels of *h*-refinement before current refinement stage
Nchild	1	Int	Number of child elements per parent elements (e.g., 4 for quadrilateral elements, 8 for brick elements; may also be 3, 5, or 9 for quadrilateral elements with h_t-refinement)
NunRef	1	Int	Number of child elements requesting unrefinement
Parent	1	Int	Element index (i.e., column number) of parent element
Pnew	1	Int	Intrinsic polynomial order of element after current refinement stage
Pold	1	Int	Intrinsic polynomial order of element before current refinement stage
pRef	1	Int	Element *p*-refinement flag: qTrue $\Longrightarrow$ refine qFalse $\Longrightarrow$ do not refine
Status	1	Int	Element refinement status: qActive or qInact
UniRef	1	Int	Uniform refinement flag: qFalse or qTrue

15.2.2.7 LRT — Line Refinement Table

An LRT is similar to an ERT, but corresponds to element lines, not elements; thus, there is only one LRT for each given mesh, typically called LINE.REFINEMENT...*mesh*. This is created and updated by the refinement processors (REF*i*). The number of columns in an LRT is equal to the total number of element lines defined for the problem. Element lines which are common to more than one element are not repeated in an LRT. When an element subdivides, the lines attached to the element also subdivide, creating new child lines, which are added to the table without deleting the original parent lines. An LRT tracks parent/child relations between element lines, in the same way as an ERT performs this function for elements.

The attributes stored in an LRT are listed in Table 15.2-26. They serve a two-fold purpose. First, parameters such as *Hnew*, *Hold*, *Pnew*, *Pold*, represent line refinement indicators, and facilitate the definition of element refinement indicators via a two-pass process. For example, the element *h*-refinement patterns are triggered by first setting the line refinement indicator *Hnew*. During refinement *Hold* is updated, when *Hold* equals *Hnew* no more refinement is needed. The second function of the LRT is to use the embedded parent/child relations between lines to facilitate the construction of interelement constraints for refinement processors that employ h_c-refinement.

Table 15.2-26 Line Refinement Table (LRT)

Attribute	Line 1	...	Line Nline
Child (Nchild)	Child $(Nchild_1)_1$	...	Child $(Nchild_{Nline})_{Nline}$
Hnew	$Hnew_1$	...	$Hnew_{Nline}$
Hold	$Hold_1$	...	$Hold_{Nline}$
Nchild	$Nchild_1$	...	$Nchild_{Nline}$
NelAtt	$NelAtt_1$	...	$NelAtt_{Nline}$
NelRef	$NelRef_1$	...	$NelRef_{Nline}$
Nnodes	$Nnodes_1$	...	$Nnodes_{Nline}$
Nodes (Nnodes)	Nodes $(Nnodes_1)_1$	...	Nodes $(Nnodes_{Nline})_{Nline}$
Parent	$Parent_1$	...	$Parent_{Nline}$
Pnew	$Pnew_1$	...	$Pnew_{Nline}$
Pold	$Pold_1$	...	$Pold_{Nline}$
Status	$Status_1$	...	$Status_{Nline}$

Columns in Table 15.2-26 range over the total number of model lines, *Nline*, including both active and inactive lines. The attributes *Nchild* and *Nnodes* indicate that the dimension of attributes *Child* and *Nodes* may vary from line to line. The maximum values of *Nchild* and *Nnodes* are stored in the CSM data object as attributes *MLchld* and *Mnline*. Attributes (rows) of an LRT data object are described in Table 15.2-27.

Table 15.2-27 LRT Attribute Descriptions

Attribute	Length	Type	Description
Child	Nchild	Int	Line (column) numbers of child lines
Hnew	1	Int	Number of levels of line *h*-refinement required
Hold	1	Int	Number of levels of line *h*-refinement before current refinement stage
Nchild	1	Int	Number of child lines
NelAtt	1	Int	Number of elements attached to line
NelRef	1	Int	Number of elements requesting refinement on line
Nnodes	1	Int	Number of nodes on line
Nodes	Nnodes	Int	Node numbers defining line
Parent	1	Int	Line number of parent line
Pnew	1	Int	Polynomial order of line after current refinement stage
Pold	1	Int	Polynomial order of line before current refinement stage
Status	1	Int	Line processing status: qActive or qInact

15.2.2.8 NDT — Nodal DOF Table

Data objects belonging to the Nodal DOF Table (NDT) class contain information about nodal freedoms (DOFs) and multi-point constraints. This includes DOF state information (e.g., fixed, free, specified, etc.), pointers representing equation numbers in an assembled system, pointers to multi-point constraint relations, and the actual data defining these constraint relations. An NDT data object is currently created and updated by processor COP.

The logical organization of a Nodal DOF Table (NDT) data object is arranged in two tables. The first table contains nodal DOF state and pointer information while the second table contains multi-point constraint information. The logical structure of the nodal DOF state/pointer information is shown in Table 15.2-28:

Table 15.2-28 Nodal DOF Table (NDT)

Attribute	Attribute	...	Node Nnode
Ptrs (Ndofn)	Ptrs $(Ndofn)_1$	...	Ptrs $(Ndofn)_{Nnode}$
State (Ndofn)	State $(Ndofn)_1$	...	State $(Ndofn)_{Nnode}$
Status	$Status_1$	...	$Status_{Nnode}$

where the columns range over the total number of nodes in the model. Attributes (rows) of an NDT are described in Table 15.2-29:

Table 15.2-29 NDT Attribute Descriptions

Attribute	Length	Type	Description
Ptrs	Ndofn	Int	Pointers to equations or multipoint constraint relations
State	Ndofn	Int	Nodal DOF state value: qFree, qMPC, qSPCnz, or qSPCz
Status	1	Int	Nodal status flag: qActive ⟹ node is activeq qInact ⟹ node is inactive

15.2.3 Non-HDB Data Structures

There are several data structures that are not currently covered under the HDB Data Object organizational umbrella, notably, system matrices (e.g., an assembled stiffness matrix) packaged in three formats: (i) compact column, (ii) compact row, and (iii) skyline. Following are examples of the data structure components containing the necessary information to describe a system matrix organized in each of the three formats. Also discussed is the physical storage of the system matrix data components as records within a single GAL dataset. Presented here are default record names, record attributes (i.e., datum type and dimensions), and specific record contents.

15.2.3.1 Compact Column Data Structure (COMPACT Format)

The COMPACT format for compact column storage of the upper triangular part of an assembled (symmetric) system matrix contains the five records shown in Table 15.2-30.

Table 15.2-30 COMPACT Formatted System-Matrix Data Structure

Record	Length	Type	Description
CONTENTS.1	20	Char	"COMPACT", self-description character string
DIAGONALS.1	N_{eq}	Float	The N_{eq} assembled system matrix diagonal entries
LOCATIONS.1	N_{vals}	Int	The ith entry of the LOCATIONS record indicates the row location (in the assembled system matrix) for the ith entry in the N_{vals}-entry VALUES record
POINTERS.1	N_{eq}	Int	The ith entry P_i of the POINTERS record indicates the end-of-information position within the LOCATIONS and VALUES records for column i of the assembled system matrix; column i of the assembled matrix has $P_i - P_{i-1}$ off-diagonal entries ($2 \le i \le N_{eq}$)
VALUES.1	N_{vals}	Float	Off-diagonal entries of the assembled system matrix

Example 15.2-1

$$\begin{bmatrix} 1.1 & 1.2 & 0.0 & 0.0 & 1.5 & 0.0 \\ & 2.2 & 2.3 & 2.4 & 0.0 & 2.6 \\ & & 3.3 & 0.0 & 3.5 & 3.6 \\ & & & 4.4 & 0.0 & 0.0 \\ & & & & 5.5 & 0.0 \\ & & & & & 6.6 \end{bmatrix}$$

This system matrix gives the COMPACT-formatted dataset records shown below.

Record	Record Contents
CONTENTS.1	COMPACT
DIAGONALS.1	1.1 2.2 3.3 4.4 5.5 6.6
POINTERS.1	0 1 2 3 5 7
LOCATIONS.1	1 2 2 1 3 2 3
VALUES.1	1.2 2.3 2.4 1.5 3.5 2.6 3.6

15.2.3.2 Compact Row Data Structure (COMPAXX Format)

The "COMPAXX" format for compact row storage of the upper triangular part of an assembled (symmetric) system matrix contains the six records shown in Table 15.2-31.

Table 15.2-31 COMPAXX Formatted System-Matrix Data Structure

Record	Length	Type	Description
CONTENTS.1	20	Char	"COMPAXX", self-description character string
COEFS.1	N_{vals}	Float	Off-diagonal entries of the assembled system matrix
COLLTH.1	N_{eq}	Int	Number of off-diagonal entries in each row of the assembled system matrix
COLPTR.1	$N_{eq}+1$	Int	The ith entry P_i of COLPTR indicates the beginning-of-information position for row i (of the assembled system matrix) in the COEFS and ROWS records; row i of the assembled matrix has P_{i+1} - P_i off-diagonal entries
DIAG.1	N_{eq}	Float	The N_{eq} assembled system matrix diagonal entries
ROWS.1	N_{vals}	Int	The ith ROWS entry is the column location (in the assembled matrix) for entry i of the COEFS record

Example 15.2-2

$$\begin{bmatrix} 1.1 & 1.2 & 0.0 & 0.0 & 1.5 & 0.0 \\ & 2.2 & 2.3 & 2.4 & 0.0 & 2.6 \\ & & 3.3 & 0.0 & 3.5 & 3.6 \\ & & & 4.4 & 0.0 & 0.0 \\ & & & & 5.5 & 0.0 \\ & & & & & 6.6 \end{bmatrix}$$

This system matrix gives the COMPAXX-formatted dataset records shown below.

Record	Record Contents
CONTENTS.1	COMPAXX
DIAG.1	1.1 2.2 3.3 4.4 5.5 6.6
COLLTH.1	2 3 2 0 0 0
COLPTR.1	1 3 6 8 8 8 8
ROWS.1	2 5 3 4 6 5 6
COEFS.1	1.2 1.5 2.3 2.4 2.6 3.5 3.6

15.2.3.3 Skyline Data Structure (SKY_MATRIX Format)

The SKY_MATRIX format for column storage of the upper triangular part of an assembled (symmetric) system matrix contains the four records shown in Table 15.2-32.

Table 15.2-32 SKY_MATRIX Formatted System-Matrix Data Structure

Record	Length	Type	Description
CONTENTS.1	20	Char	"SKY_MATRIX", self-description character string
DIAG_POINTER.1	N_{eq}	Int	The magnitude of entry *i* in the DIAG_POINTER record indicates the location (in the MATRIX record) of the diagonal entry for equation *i* of the assembled system matrix; the sign indicates the constraint status for equation *i*: positive if equation *i* is not constrained; negative if equation *i* is *SPCz*- or *SPCnz*-constrained
MATRIX.1	N_{vals}	Float	Assembled system matrix entries; the active part of each column of the upper-triangular half of the matrix is stored sequentially here (incl. zeros)
TYPE.1	20	Char	Status (i.e., "FACTORED" or "UNFACTORED")

Example 15.2-3

$$\begin{bmatrix} 1.1 & 1.2 & 0.0 & 0.0 & 1.5 & 0.0 \\ & 2.2 & 2.3 & 2.4 & 0.0 & 2.6 \\ & & 3.3 & 0.0 & 3.5 & 3.6 \\ & & & 4.4 & 0.0 & 0.0 \\ & & & & 5.5 & 0.0 \\ & & & & & 6.6 \end{bmatrix}$$

This system matrix gives the COMPACT-formatted dataset records shown below:

Record	Record Contents
CONTENTS.1	SKY_MATRIX
DIAG_POINTER.1	1 3 5 8 13 18
VALUES.1	1.1 1.2 2.2 2.3 3.3 2.4 0.0 4.4 1.5 0.0 3.5 0.0 5.5 2.6 3.6 0.0 0.0 6.6
TYPE.1	UNFACTORED

15.2.4 References

[1] Stanley, G. and Swenson, L., *HDB: Object-Oriented Database Utilities for COMET-AR*, NASA Computational Structural Mechanics (CSM) Contract Report, August 1992.

[2] Felippa, C., Regelbrugge, M., and Wright, M., *The Computational Structural Mechanics Testbed Architecture, Volume IV: The Global Database Manager GAL-DBM*, NASA CR-178387, 1989.

[3] Stehlin, B., *DB/MEM: Generic Database Utilities for the COMET-AR Testbed*, NASA Computational Structural Mechanics (CSM) Contract Report, May 1992.

15.3 Database Access

In this section we point to the mechanisms which COMET-AR processor and procedure user/ developers may employ to access the database created in the context of adaptive refinement.

15.3.1 Processor-Level Database Access

All of the data objects described in the previous section may be conveniently accessed at the COMET-AR processor (i.e., FORTRAN) level by calling on HDB object-oriented high-level data utilities. These are discussed fully in the HDB Manual [1]. Each data object has its own set of HDB utilities to perform the basic functions of getting/putting data from/into the database, printing, copying, etc. Additionally, some data objects have tailor-made utilities to perform special-purpose functions. For example, EDT class objects have a utility called EDTnext which gets the index of the next active element; an LRT class object has a utility called LRTvert which gets the vertex node numbers of a line given the line number, etc. Consult Reference [1] before employing the HDB utilities to build (or retrofit) Testbed processors.

15.3.2 Procedure-Level Database Access

At the COMET-AR command procedure level (batch or interactive), the user may get information from the database via the CLIP *G2M (GAL to Macro) directive. This requires a detailed knowledge of the physical structure of the dataset in terms of record organization, a level of detail which is not required at the processor level. To alleviate this burden, we have developed a utility procedure, called CSMget and used within the ES Procedure, which may be used to get attributes from a CSM (Complete Summary of the Model) data object in a logical (versus physical) manner. This utility procedure is like an object-oriented version of the *G2M directive. It is incomplete, however, and other utility procedures will be required to access information from other data objects.

To print all or part of a data object in a meaningful fashion, either from a COMET-AR procedure or interactively, the user may invoke the HDB print processor described in Section 14.3, *Database Print Utility.*

15.3.3 References

[1] Stanley, G. and Swenson, L., *HDB: Object-Oriented Database Utilities for COMET-AR*, NASA Computational Structural Mechanics (CSM) Contract Report, August 1992.

15.4 Database Organization and Evolution

This section presents a discussion of the organization and evolution of a COMET-AR database during an AR analysis. This includes the distribution of data sets among data libraries (i.e., host files), and the growth of both data libraries and datasets as the mesh (and solution) is iteratively updated during AR. In addition, a summary of datasets (and their class membership) is presented where the names of processors that possibly create or use each dataset are given.

15.4.1 Data Libraries

A COMET-AR database is usually not stored on a single GAL data library during adaptive refinement. Instead, the database is divided into 3 files: (i) *Case*.DBC; (ii) *Case*.DBE; and (iii) *Case*.DBS. The .DBC file may be viewed as the computational data library. It contains everything except element and system matrices. The .DBE file contains only element matrices (e.g., stiffness and mass). The .DBS file contains only assembled structural system matrices. During adaptive refinement, the evolution of these libraries is shown in Figure 15.4-1.

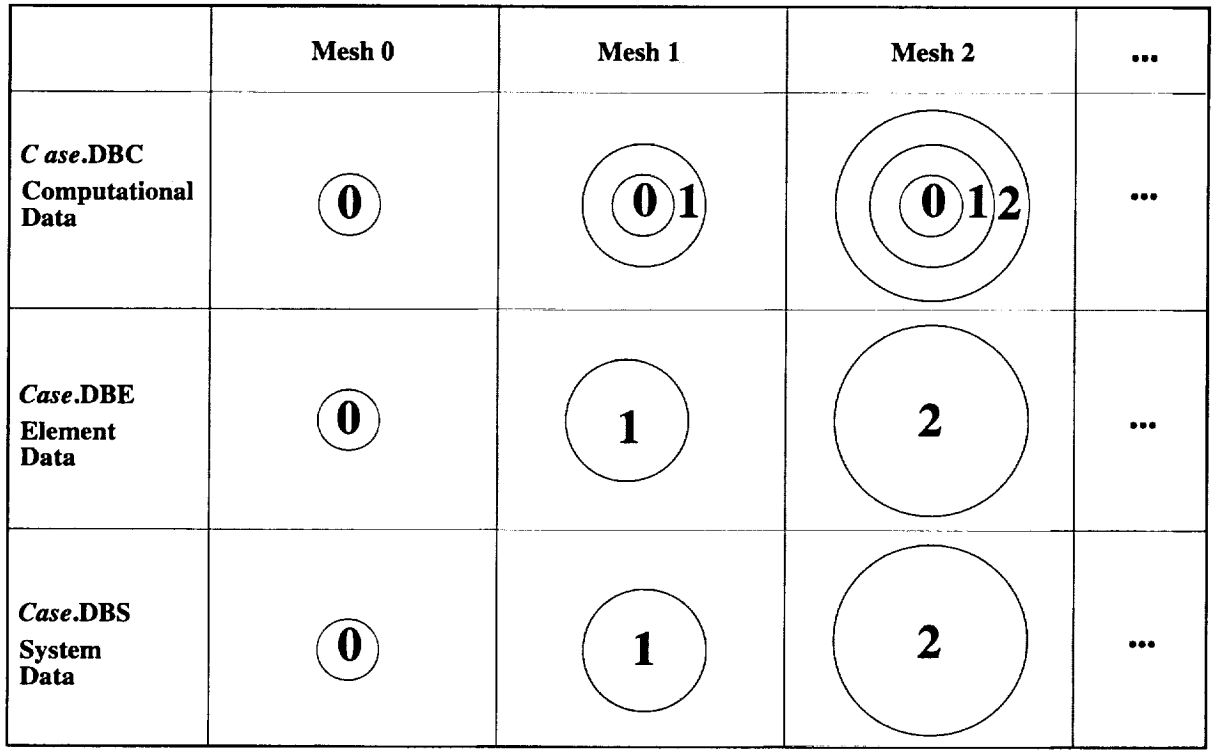


Figure 15.4-1 Evolution of COMET-AR Data Libraries during Adaptive Refinement

As indicated in Figure 15.4-1, the .DBC file continues to grow with each new mesh iteration, saving all model and solution data for every mesh analyzed. In contrast, the .DBE and .DBS files are deleted and re-created at the beginning of each new mesh iteration. This is because the element and system matrices are relatively large and are typically not needed once the mesh has been updated.

There are exceptions to this last rule, which may be exploited in the future. For example, during adaptive *h*-refinement of a linear analysis, it may be advantageous to retain the element matrix file (.DBE) and simply extend it as new elements are added so that only the matrices for the newly added elements need to be reformed. Similarly, for adaptive *p*-refinement with hierarchical-type *p* elements, the element matrix file may be updated rather than re-created. Even the assembled matrices in the .DBS file may be handled this way (for linear analysis) if partial factoring or iterative algorithms are used to solve the linear equation system.

The evolution of data libraries during adaptive refinement is currently managed by the AR control procedure, AR_CONTROL, described in Chapter 4, *Adaptive Solution Procedures*.

15.4.2 Dataset Evolution

There are two different ways to manage datasets during adaptive refinement: (i) extension and (ii) re-creation. With the extension approach, datasets are extended as new elements and nodes are added. For *h*-refinement, this is simply a matter of adding element and nodal records to a dataset, without changing the length of each record. For *p*-refinement, either the length of element records must be extended or the initial element record length must be reserved large enough to accommodate the maximum level of *p* used in the problem. For *h/p*-refinement, extensions in both record length and record number are required. The situation is illustrated in Figure 15.4-2.

With the re-creation approach entirely new datasets are created for each new mesh, leaving the original datasets (for the old mesh) intact. For *h*-refinement, this is accomplished by first copying the old dataset to a new one (via the HDB data object copy utilities), and then adding records. For *p*-refinement, the datasets are simultaneously copied and expanded in length (again via the HDB copy utilities). For *h/p*-refinement, the datasets are first copied/expanded, and then extended by records. For uniform *p*-refinement, while the record lengths must be increased, all records remain the same size for a given element type. This is true for both the extension and the re-creation approaches.

With the extension approach, the size of the database is minimized (an advantage). However, it becomes difficult to reconstruct earlier meshes for post-processing and/or restart purposes, and it is very difficult to manage *p*-refinement (or *h/p*-refinement) without reserving maximum-*p* record sizes at the outset (both liabilities). There are also a number of datasets, particularly solution datasets, which must be re-created, as the values are different for the same nodes and elements from one mesh to the next.

A similar trade-off exists for the re-creation approach. While the size of the database grows more rapidly than with the extension approach (a liability), all previous meshes are instantly available for post-processing and/or re-starts, and *p*- and *h/p*-refinement present no additional difficulties as each new element dataset may be sized for precisely what is needed in the new mesh (both advantages).

In light of the above trade-offs, the re-creation approach is the one adopted by the refinement processors currently implemented in COMET-AR (e.g., REF1). Both approaches are potentially available by invoking the appropriate option at the HDB database utility level.

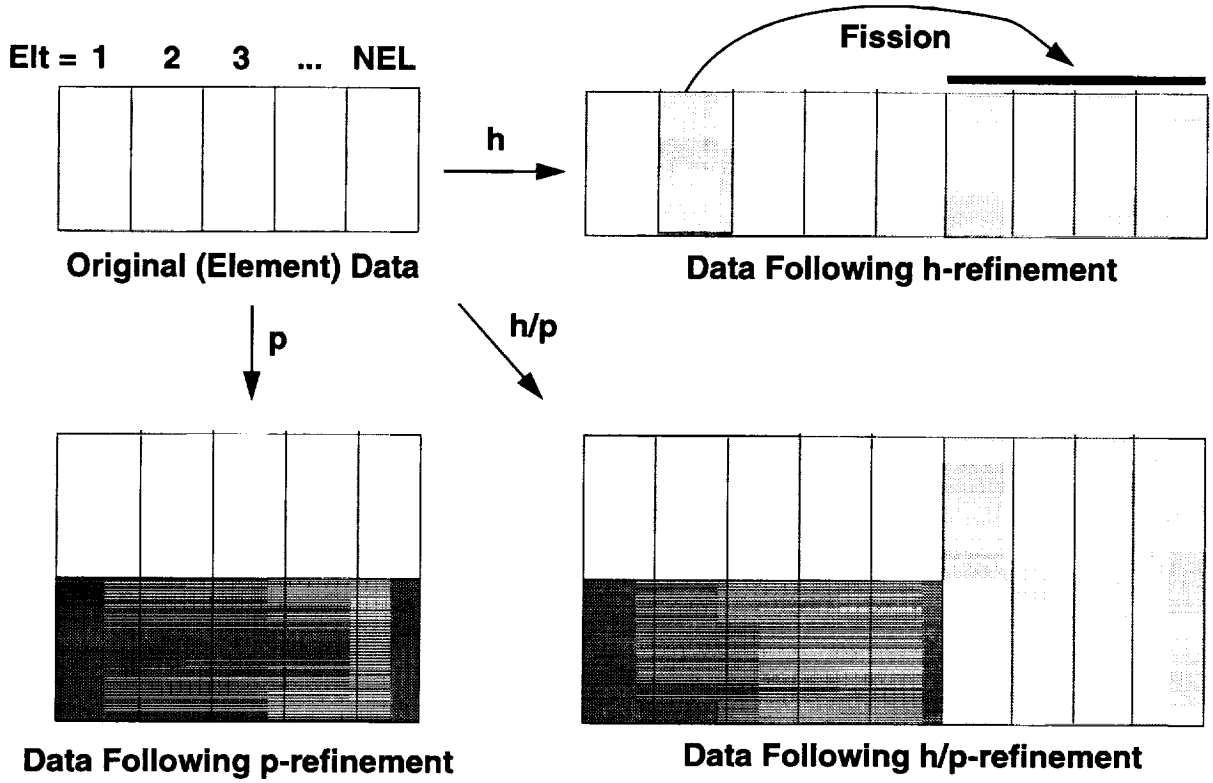


Figure 15.4-2 Evolution of Datasets during Adaptive Refinement

15.4.3 Dataset Creation and Usage

During the analysis process, numerous datasets are created and used by COMET-AR processors. Tables 15.4-1 through 15.4-6 list all High-level Database (HDB, see Reference [1]) datasets by generic name and summarize which processors read from or write to them. Processors that may create a dataset are tagged with an asterisk.

Table 15.4-1 Processor Usage of Summary Data Object Datasets

Class	Generic Dataset Name	Processor Name	
		Input Usage	Output Usage
ANS	ANALYSIS.SUMMARY	—Not Currently Used—	—Not Currently Used—
ARS	REFINEMENT.SUMMARY	—Not Currently Used—	—Not Currently Used—
CSM	CSM.SUMMARY	—All Processors—	—All Processors—

Table 15.4-2 Processor Usage of Element Data Object Dataset Names

Class	Generic Dataset Name	Processor Name	
		Input Usage	Output Usage
EAT	*EltName.AttName*	ASMs, SKYs	ASMs*, ES*i*, SKYs
EDT	*EltName*.DEFINITION	ASM, ASMs, ERR*i*, ES*i*, REF1	REDO*, REF1*
EET	*EltName*.ERROR	ERRa, REF1	ERR*i*
EFT	*EltName*.FABRICATION	ERR2, ERR6, ES*i*, GCP	ES*i*, REDO*
EGT	*EltName*.GEOMETRY	REF1	REDO*, REF1*
EIT	*EltName*.INTERPOLATION	ERR*i*, REF1	ES*i**
ELT	*EltName*.LOAD	ES*i*, REF1	ES*i**, REF1*
EMT	*EltName.MatName*	ASM, ASMs	ES*i**
ERT	*EltName*.REFINEMENT	REF1	REF1*
EST	*EltName.StrName*	ERR*i*	ES*i**

Table 15.4-3 Usage of Nodal Data Object Dataset Names

Class	Generic Dataset Name	Processor Name	
		Input Usage	Output Usage
NAT	NODAL.*AttName*	ES*i*	ES*i**
NCT	NODAL.COORDINATE	—All Processors—	REDO*, REF1*
NDT	NODAL.DOF	ASM, ASMs, COP, REF1	COP*, REF1*
NOT	NODAL.ORDER	COP	RENO*, RSEQ*
NTT	NODAL.TRANSFORMATION	ASMs, ES*i*, REF1, TRIAD, SKYs	REDO*, REF1*, TRIAD*
NVT	NODAL.*VecName*	ASM, ASMs, ES*i*, REF1	COP*, REF1

Table 15.4-4 Processor Usage of Line Data Object Dataset Names

Class	Generic Dataset Name	Processor Name	
		Input Usage	Output Usage
LRT	LINE.REFINEMENT	REF1	REF1*

Table 15.4-5 Processor Usage of Surface Data Object Dataset Names

Class	Generic Dataset Name	Processor Name	
		Input Usage	Output Usage
SRT	SURFACE.REFINEMENT	REF1	REF1*

Table 15.4-6 Processor Usage of Matrix/Vector Data Object Dataset Names

Class	Generic Dataset Name	Processor Name	
		Input Usage	Output Usage
SMT	SYSTEM.*MatName*	SKYs	ASMs*
SVT	SYSTEM.*VecName*	ITER, SKY, SKYs	ASM*, ASMs*, ITER*, SKY*, SKYs*

15.4.4 References

[1] Stanley, G. and Swenson, L., *HDB: Object-Oriented Database Utilities for COMET-AR*, NASA Computational Structural Mechanics (CSM) Contract Report, August 1992.

Part V

Solid Model Interface

(SMI)

In this part of the COMET-AR User's Manual, we describe the solid model interface to COMET-AR, i.e., the various ways in which the underlying geometry of a COMET-AR model can be represented. This includes the conventional approach of using the initial finite-element model as a solid model representation, and a user-written solid model approach that is more accurate for purposes of adaptive mesh refinement, but places more of a burden on the user. In the future it is planned to integrate COMET-AR with a Computer-Aided Design (CAD) system, so that an accurate solid-model description is maintained throughout an adaptively-refined analysis.

16 Solid Model Interface (SMI)

16.1 Overview

In this chapter, we describe the solid model interface (SMI) options available with adaptive refinement. The SMI is the link between the discrete finite element model and the underlying continuous (i.e., solid model) description of the structure. As indicated in Table 16.1-1, two SMI options are described in this chapter: discrete (approximate), and user-written (exact).

Table 16.1-1 Outline of Chapter 16: Solid Model Interface (SMI)

Section	Topic
16.2	SMI Options
16.3	The Discrete (Approximate) SMI Option
16.4	The User-Written (Exact) SMI Option

16.2 Solid Model Interface (SMI) Options

Two solid model interface options are provided in COMET–AR: discrete and user-written. The discrete SMI employs the initial finite element model as the exact solid model description, and saves the user the trouble of defining any additional information (beyond the usual finite element model input). In contrast, the user-written SMI relies on the user to provide a continuous description of the exact solid model via user-written subroutines and some simple links to the initial finite element model.

Thus, the user has the choice of either defining a sufficiently refined initial finite element model to accurately represent the geometry, loads, material properties, and boundary conditions; or using an arbitrarily coarse initial finite element model, but describing all of the functional variations in these quantities via user-written subroutines.

Both the discrete and user-written SMIs are embedded in a generic SMI cover routine (called SMshlxx), which is employed by all standard mesh refinement processors (i.e., REF*i*). The implementation of the various SMI options within REF*i* is shown schematically in Figure 16.2-1.

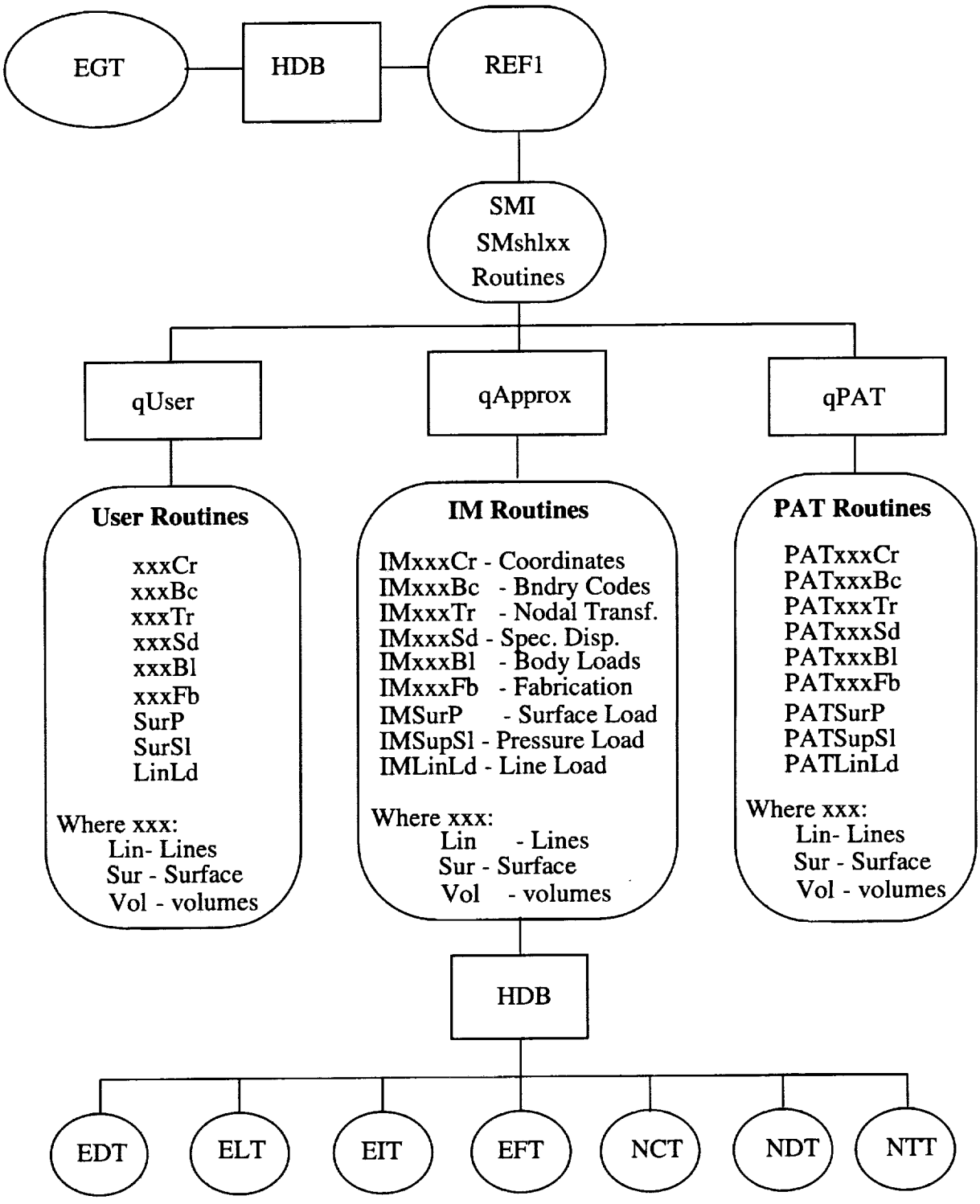


Figure 16.2-1 SMI Implementation Within Mesh Refinement Processors

16.3 The Discrete SMI Option

The user may select the discrete SMI option by setting the /SOLID_MODEL qualifier in the generic element processor's DEFINE ELEMENTS command to DISCRETE:

```
[RUN ES1
   RESET ELEMENT_TYPE = EX97
   DEFINE ELEMENTS /SOLID_MODEL = DISCRETE
   ELEMENTS 1 NODES = n1, n2, n3, n4,...
   ...
```

See Chapter 7, *Element Processors*, for a complete description of processor ES*i*'s DEFINE ELEMENTS command. /SOLID_MODEL = DISCRETE is currently the default option.

The initial finite element model is interpreted as the solid model, and all subsequent adaptive refinements are performed by interpolating geometry, material properties, loads, and boundary conditions from this discrete model, element by element.

While this SMI option is extremely straightforward it can lead to erroneous AR results, converging to the wrong solution if the initial discrete model is not sufficiently detailed to pick up all important variations in structural geometry, properties, etc. This may be difficult to ascertain a priori in many cases. The use of potentially curved, higher-order elements (i.e., quadratic and higher) is recommended in conjunction with the discrete SMI, as this can dramatically increase the geometric accuracy of the initial finite element model and alleviate this pitfall.

16.4 The User-Written SMI Option

To employ the user-written (exact) solid-model interface option, the user must first make the links between the initial model and the exact model during initial model generation, and then write a set of subroutines which will enable a refinement processor (REF*i*) to update the mesh and remain faithful to the user's exact model. These user-written routines must be linked into the appropriate REF*i* processor executable.

The ingredients for both of these steps are described in the following subsections.

16.4.1 Initial Model Generation

The user may select the user-written SMI option by setting the /SOLID_MODEL qualifier in a generic element (ES*i*) processor's DEFINE ELEMENTS command to USER:

```
RUN ES1

   RESET ELEMENT_TYPE = EX97

   DEFINE ELEMENTS /SOLID_MODEL = USER

   RESET SURFACE = S_1

   ELEMENTS 1  NODES = n1, n2, n3, n4, ...  LINES = l1, l2, ...
```

The RESET SURFACE command and the "LINES = l1, l2,..." subcommands will be explained below by example. They establish the links between the initial finite element model and the user's conception of the solid model.

Figure 16.4-1 provides an example of an initial finite element model for a composite cylindrical panel with circular cutout (also known as "Knight's Panel" problem). It consists of one surface, S1, and six lines: L_1, L_2, L_3, L_4, L_5 and L_6. The lines represent the four boundaries of the panel plus the internal boundary of the hole. The external boundaries have the displacement conditions indicated below:

Line Number	Line Boundary Conditions
1	Zero: D_1, D_2, $Theta_1$, $Theta_2$, $Theta_3$
2	Zero: D_1, D_2, D_3, $Theta_1$, $Theta_2$, $Theta_3$
3	Zero: D_1, $Theta_1$, $Theta_3$
4	Zero: D_1, $Theta_1$, $Theta_3$
5	Zero: $Theta_3$
6	Zero: $Theta_3$

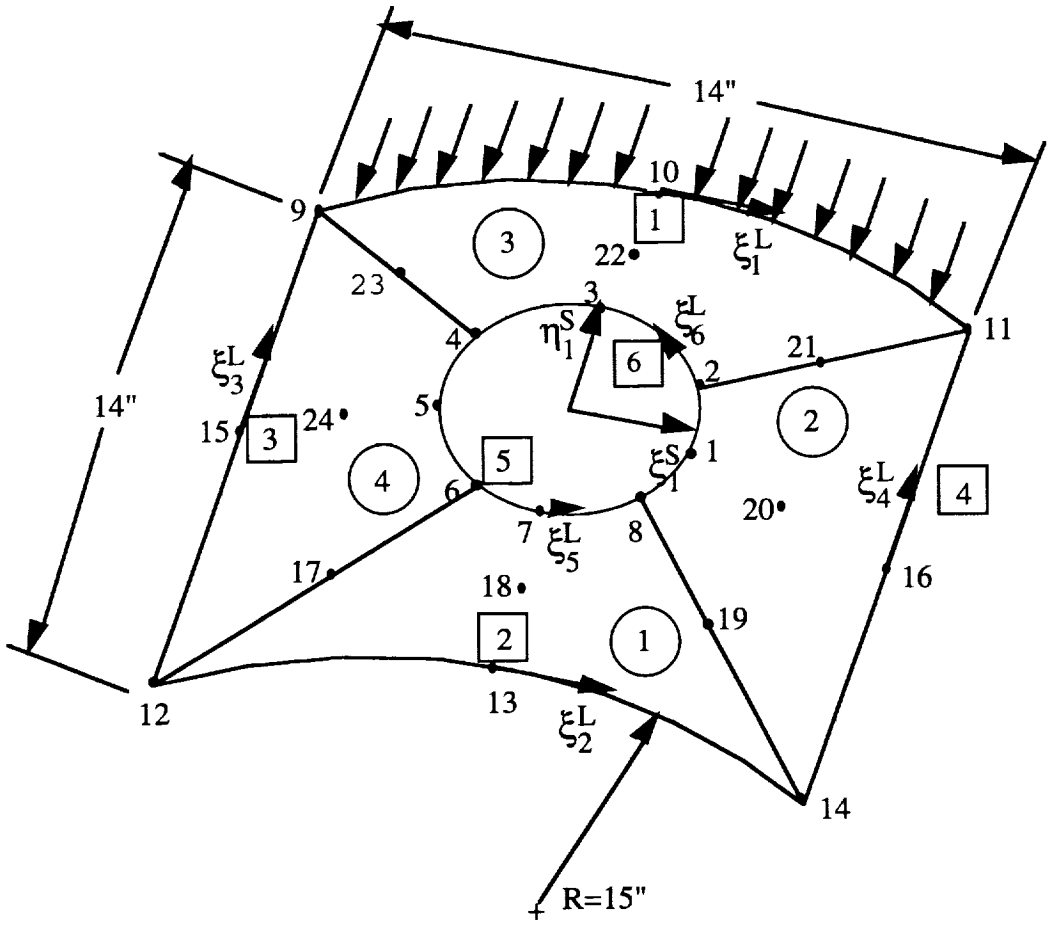


Figure 16.4-1 User-Written Solid Model Description of Knight's Panel

For this problem, the appropriate initial element definition subcommands follow.

```
DEFINE ELEMENTS /SOLID_MODEL = USER

   RESET SURFACE = 1

   ELEMENT 1 NODES = 12,14,8,6,13,19,7,17,18  LINES = 2,0,5,0

   ELEMENT 2 NODES = 14,11,2,8,16,21,1,19,20  LINES = 4,0,6,0

   ELEMENT 3 NODES = 11, 9,4,2,10,23,3,21,22  LINES = 1,0,6,0

   ELEMENT 4 NODES =  9,12,6,4,15,17,5,23,24  LINES = 3,0,5,0
```

The solid model lines are associated with element sides via the LINES phrase. Element sides that do not lie on solid model lines are assigned a line number of zero, while those which do are assigned the solid model line number. Similarly, for 3-D solid elements there is both a LINES and a SURFACES phrase and a RESET VOLUME command.

To perform adaptive refinement using this model as the underlying solid model, the user must also provide a set of user-written subroutines describing each of the solid model's properties (geometry, loads, boundary conditions, etc.) as functions of the intrinsic surface parameters (ξ,η) for each distinct solid-model surface, and as a function of the intrinsic line parameter (ξ) for each distinct solid model line. These routines are described in the following subsections.

16.4.2 User-Written SMI Routines

16.4.2.1 General Description and Summary

The user-written subroutines required to perform adaptive refinement employing the user-written SMI are summarized in this section. There are subroutines describing solid model properties (such as global coordinates, transformations, material properties, loads, and boundary conditions) for line, surface, and volume entities. Mathematically, these subroutines use the following functions:

Along lines:

$$X = X(\xi_i^L)$$

Along surfaces:

$$X = X(\xi_i^S, \eta_i^S)$$

Within volumes:

$$X = X(\xi_i^V, \eta_i^V, \zeta_i^V)$$

where ξ, η, and ζ are intrinsic line, surface, and volume coordinates for each geometric entity (i.e., line, surface, volume). Each coordinate ranges between –1 and +1 along that entity.

The complete set of user-written subroutines required for the most general case are listed in Table 16.4-1.

Table 16.4-1 Summary of User-Written SMI Subroutines

Generic Name	Specific Versions (smi)			Purpose of Subroutine
	LIN	SUR	VOL	
*smi*BCS	✔	✔	✔	Define boundary conditions (DOF states)
*smi*BLD	✔	✔	✔	Define body loads
*smi*CRD	✔	✔	✔	Define global coordinates
*smi*FAB	✔	✔	✔	Define material fabrication associations
*smi*LLD	✔	N/A	N/A	Define line loads

Table 16.4-1 Summary of User-Written SMI Subroutines (Continued)

Generic Name	Specific Versions (smi)			Purpose of Subroutine
	LIN	SUR	VOL	
*smi*PLD	N/A	✔	N/A	Define pressure loads
*smi*SLD	N/A	✔	N/A	Define general surface loads
*smi*SPD	✔	✔	✔	Define specified displacements
*smi*TRF	✔	✔	✔	Define global/computational transformations

In practice, the letters "smi" in the subroutine names in Table 16.4-1 are replaced by LIN, SUR, or VOL for line, surface, or volume entities, as appropriate. In most cases only a subset of these subroutines must be written. For example, for a problem involving only 1-D elements (e.g., beams), only LIN subroutines are needed, and while all of the geometric and material entry points must be written, only those load entry points (LINLLD, SURSLD, etc.) which are relevant to the problem must be written.

Before performing AR with the user-written SMI option, the user must link the above subroutines into the appropriate mesh refinement processor (REF*i*). See the REF*i* "makefile" for details.

16.4.2.2 SMI Subroutine Argument Glossary

Table 16.4-2 provides a comprehensive list of arguments that appear in one or more of the user-written SMI subroutines.

Table 16.4-2 User-Written SMI Subroutine Argument Glossary

Name	Type	I/O	Description
Bcs	Integer(Ndof)	O	DOF states; Bcs(i) = { qSPCz, qFree, ... }
Blds	Float(Ndof)	O	Body load vector (force/volume)
Crds	Float(3)	O	Global coordinates: Crds(i) = x_i^g
ETA	Float	I	Second SM coordinate (η)
FabEcc	Float(NfabEcc)	O	Fabrication eccentricities
FabID	Integer	O	Fabrication ID (in FE database)
FabOri	Float(NfabOri)	I	Fabrication orientation data
FabRef	Integer	O	Fabrication reference frame (in FE database)
Lexist	Integer	O	Load existence flag: qTrue or qFalse
LinID	Integer	I	Line ID
Llds	Float(Ndof)	O	Line load vector (force/length)

Table 16.4-2 User-Written SMI Subroutine Argument Glossary (Continued)

Name	Type	I/O	Description
Ndof	Integer	I	Number of DOF per node
NfabEcc	Integer	O	Number of fabrication eccentricities
NfabOri	Integer	I	Number of fabrication orientation data items
Plds	Float	O	Pressure load (force/area)
Slds	Float(Ndof)	O	Surface load (traction) vector (force/area)
Spds	Float(Ndof)	O	Specified displacement vector
Status	Integer	I	Subroutine return status (**qOK** ⇒ no errors)
SurID	Integer	I	Surface ID
Trfs	Float (3,3)	O	Transformations: Trfs(i,j) = T_{ij}^{GC}
XSI	Float	I	First SM coordinate (ξ)
VolID	Integer	I	Volume ID
ZETA	Float	I	Third SM coordinate (ζ)

In the following sections the calling sequences for each of the user-written SMI subroutines listed in Table 16.4-1, and involving the arguments listed in Table 16.4-2, are presented. In the following calling sequences the prefix symbol < denotes an input argument, the symbol > denotes an output argument, and the symbol <> denotes an argument that is used both as input and output.

16.4.2.3 BCS (Boundary-Condition) SMI Utilities

Name	Calling Sequence
LinBCS	(<LinID, >BCs, <Ndofe, >Status)
SurBCS	(<SurID, >BCs, <Ndofe, >Status)
VolBCS	(<VolID, >BCs, <Ndofe, >Status)

16.4.2.4 BLD (Body-Load) SMI Utilities

Name	Calling Sequence
LinBLD	(<LinID, <XSI, >Blds, <Ndof, >Lexist, >Status)
SurBLD	(<SurID, <XSI, <ETA, >Blds, <Ndof, >Lexist, >Status)
VolBLD	(<VolID, <XSI, <ETA, <ZETA, >Blds, <Ndof, >Lexist, >Status)

16.4.2.5 CRD (Coordinate) SMI Utilities

Name	Calling Sequence
LinCRD	(<LinID, <XSI, >Crds, >Status)
SurCRD	(<SurID, <XSI, <ETA, >Crds, >Status)
VolCRD	(<VolID, <XSI, <ETA, <ZETA, >Crds, >Status)

16.4.2.6 FAB (Fabrication) SMI Utilities

Name	Calling Sequence
LinFAB	(<LinID, <XSI, >FabID, >FabEcc, >FabRef, >NfabEcc, >FabOri, >NFabOri, >Status)
SurFAB	(<SurID, <XSI, <ETA, >FabID, >FabEcc, >FabRef, >NfabEcc, >FabOri, >NFabOri, >Status)
VolFAB	(<VolID, <XSI, <ETA, <ZETA, >FabID, >FabEcc, >FabRef, >NfabEcc, >FabOri, >NFabOri, >Status)

16.4.2.7 LLD (Line-Load) SMI Utilities

Name	Calling Sequence
LinLLD	(<LinID, <XSI, >Llds, >Lexist, >Status)

16.4.2.8 PLD (Pressure-Load) SMI Utilities

Name	Calling Sequence
SurPLD	(<SurID, <XSI, <ETA, >Slds, <Ndof, >Lexist, >Status)

16.4.2.9 SLD (Surface-Load) SMI Utilities

Name	Calling Sequence
LinSLD	(<LinID, <XSI, >Slds, <Ndof, >Lexist, >Status)
SurSLD	(<SurID, <XSI, <ETA, >Slds, <Ndof, >Lexist, >Status)
VolSLD	(<VolID, <XSI, <ETA, <ZETA, >Slds, <Ndof, >Lexist, >Status)

16.4.2.10 SPD (Specified-Displacement) SMI Utilities

Name	Calling Sequence
LinSPD	(<LinID, <XSI, >Spds, <Ndof, >Lexist, >Status)
SurSPD	(<SurID, <XSI, <ETA, >Spds, <Ndof, >Lexist, >Status)
VolSPD	(<VolID, <XSI, <ETA, <ZETA, >Spds, <Ndof, >Lexist, >Status)

16.4.2.11 TRF (Transformation) SMI Utilities

Name	Calling Sequence
LinTRF	(<LinID, <XSI, >Trfs, >Status)
SurTRF	(<SurID, <XSI, <ETA, >Trfs, >Status)
VolTRF	(<VolID, <XSI, <ETA, <ZETA, >Trfs, >Status)

16.4.2.12 Mesh Update Algorithm via User-Written SMI Utilities

The user may have noticed that with the user-written SMI option, the user does not provide the values of the line and surface coordinates (ξ, η, and ζ, respectively) for the nodes in the initial finite element mesh. Since these coordinates are provided as an input data by a REF*i* processor to the user-written SMI routines (via the generic SMI shell routine described earlier), the question might arise as to how they are generated, both for the nodes in the original mesh and for new nodes created during adaptive refinement.

To save the user the trouble of defining the line and surface intrinsic coordinates at each of the nodes in the initial mesh, the SMI shell uses the following algorithm to generate these intrinsic coordinates during adaptive refinement.

1) Predict the values of ξ at each node of the parent element, where the global coordinates are known as $\boldsymbol{x}$.

2) Compute the actual values of ξ corresponding to $\boldsymbol{x}$ (at each parent node) by inverting the mapping $\boldsymbol{x} = f(\xi)$ provided by the user-written subroutine LINCRD. The inverse mapping is obtained by linearizing the following "projected" version of the forward mapping:

$$\frac{\partial f}{\partial \xi_\alpha} \cdot (\boldsymbol{x} - f(\xi)) = 0 \qquad \alpha = 1, 2, \ldots, N_{dim}$$

and employing the following Newton-Raphson algorithm:

$$\left[\frac{\partial^2 f}{\partial \xi_\alpha \partial \xi_\beta} \cdot (\boldsymbol{x} - f(\xi)) - \frac{\partial f}{\partial \xi_\alpha} - \frac{\partial f}{\partial \xi_\beta}\right]_{\xi^i} \delta\xi_\beta^{i+1} = -\left.\frac{\partial f}{\partial \xi_\alpha}\right|_{\xi^i} \cdot (\boldsymbol{x} - f(\xi^i))$$

$$\xi_\beta^{i+1} = \xi_\beta^i + \delta\xi_\beta^{i+1}$$

3) Go to Step 1 and iterate until $\xi^{i+1} \approx \xi^i$

In these equations, α and β each range from 1 to the number of intrinsic dimensions, N_{dim}. The value of N_{dim} equals 1 for a curve, 2 for a surface, and 3 for a volume.

4) Once ξ is obtained at the parent element's nodes, the values of ξ at the new interior nodes engendered by refining the element are obtained by interpolation. The global coordinates (and other properties) at the new nodes are then obtained via the user-written SMI routines.

The derivatives of f appearing in Step 1 are computed numerically, via second-order finite-difference approximations:

$$\frac{\partial f}{\partial \xi_\alpha} = \frac{1}{2\varepsilon}[f(\xi_\alpha + \varepsilon) - f(\xi_\alpha - \varepsilon)]$$

$$\frac{\partial^2 f}{\partial\, \xi_a \partial \xi_\beta} = \frac{1}{2\varepsilon}\left[\frac{\partial f}{\partial \xi_\alpha}(\xi_\alpha, \xi_\beta + \varepsilon) - \frac{\partial f}{\partial \xi_\alpha}(\xi_\alpha, \xi_\beta - \varepsilon)\right]$$

The inverse mapping is from a 3-D global space to either a 1-D, 2-D, or 3-D intrinsic space. The projection of the forward mapping given as the first equation under Step 2 is necessary to handle the case where the number of intrinsic dimensions is less than 3, for example along a curved line or surface. As a fringe benefit, the same procedure may be used to compute ξ given a starting point, $\boldsymbol{x}$, which does not even lie in the intrinsic space (i.e., on the curve or surface). In this case, the value of ξ computed will correspond to the point on the curve or surface which is closest to the starting point, $\boldsymbol{x}$; that is, the orthogonality condition $\partial f/\partial\xi \cdot \boldsymbol{x} - f(\xi) = 0$ will yield the orthogonal projection of $\boldsymbol{x}$ onto that curve/surface.

The projected inverse mapping procedure is implemented in the subroutine utility package xxxProj, which is invoked via the generic SMI cover routine SMShl*xx* that is employed in all standard REF*i* processors. These utilities may also be invoked via the following calling sequence:

```
CALL xxxProj ( <xxxID, <Pt, >ξ, >η, >ζ, >Crds, <LIM, <>Status )
```

where the symbols are defined as follows.

Symbol	Definition
xxx	Lin ⟹ line projection Sur ⟹ surface projection Vol ⟹ volume projection
Pt(3)	Coordinates of the point as generated by REF1

Symbol	Definition
ξ,η,ζ	Intrinsic coordinates of the projected point; only ξ is relevant for lines; and only ξ and η are relevant for surfaces
Crds(3)	Projected point coordinates in the computational frame
LIM	Initial model logical flag: .true. $\Rightarrow$ initial model routines are used (CSM.SolMod = qApprox) .false. $\Rightarrow$ user routines are used (CSM.SolMod = qUser or qPAT)
Status	SMI status flag

16.4.3 Limitations on User-Written SMI Option

1) The user may define lines, surface, or volumes of any shape via the user-written SMI as long as every point within these solid-model entities has a unique, and invertible, mapping $\boldsymbol{x} = \boldsymbol{f}(\xi)$. Closed curves and surfaces must be subdivided into open curves and surfaces before defining them as solid model entities. For example, to model a closed cylindrical shell, break the cylindrical surface into two cylindrical surfaces (e.g., 0–180 deg., and 180–360 deg.).

2) Each element and its boundaries must be contained within a single geometrical entity. For example, a shell element must be contained within a single surface definition and each of its four edges which coincides with geometrical lines must be contained within a single line. (e.g., elements cannot be defined across geometrical surface boundaries, and element edges cannot be defined across geometrical line boundaries).

3) To increase the efficiency of the mesh projection algorithm employed for the user-written SMI, keep the range of the intrinsic line, surface, and volume coordinates (ξ,η,ζ) as close to the interval [-1,+1] as possible. This can speed up the convergence of the embedded Newton-Raphson algorithm considerably.

16.4.4 Example: User-Written SMI for the Knight's Panel Model

In this section, we demonstrate how to write a complete set of user-written solid model routines using the Knight's panel example depicted in Figure 16.4-2.

In general, a good starting point for developing a new user-written solid model routines is to obtain a copy of an existing example, such as the one discussed in this section, and to modify the example for the particular problem.

User-written solid model routines are treated, by design, as a complete solid modeler by a refinement processor and as such, they should provide the full range of solid model entry points, including those which may not be relevant to the particular problem at hand. Those entry points which are not used for modeling the particular problem can be present as simple subroutine "stubs", having all the formal arguments declared but no statements.

All solid modelers operate on a series of hierarchically ordered geometrical entities: point, curve, surface, and volume (presented in their descending hierarchical order). Entities which have a higher hierarchical position also have higher priority. If information is requested by a refinement processor at a location in the model which belongs to both a geometrical surface and a geometrical curve, the curve's higher position in the hierarchical list will give it higher priority over the surface, and information for this location will be provided based on the curve's data.

Only the curve, surface, and volume entities are relevant in the context of adaptive mesh refinement, since points can not be refined. The geometry points data must be present in the initial mesh and is not modified throughout the adaptive mesh refinement process.

A few words about the programming style used in this example are required. Each of the routine contains two distinct sections: "Declarations" and "Logic". The Declarations section contains an explicit declaration for each of the routine's arguments as well as any internal variables. A machine-dependent implicit declaration of all variables in the subroutine (e.g., the MAX blocks in lines 17-27 of Listing 1) eliminates many hard-to-find variable names typing errors. The Declarations section in each routine contains the standard COMET-AR Q-symbols file (see Reference [1] for details).

Error processing is an important aspect of programming within the COMET-AR architecture. Each routine contain a formal argument "Status" which is used for monitoring execution errors. The first executable statement in each routine verifies that there is no prior error condition set by any of the former routines. This is accomplished by a logical check comparing the value of the Status argument to the Q-symbol qOK value. If an error condition is set prior to the routine invocation, the routine should silently exit such that the error message trace-back produced by higher level routines will not be cluttered by messages from lower level routines, which cannot have any relevant information regarding the particular error condition.

If an error is detected during the normal execution of a routine, the routine is required to initiate the error processing mechanism by calling a standard entry point named "ERR" to set the error condition. The ERR routine has three formal arguments: the name of the calling routine, an error message, and the Status argument. The ERR routine will automatically set the execution error condition and initiate the error trace-back mechanism.

Any messages to the user should be printed using the PRT*i* printing entry points to maintain a consistent format of messages. This example may be used as a template for writing messages.

In the following subsections, each of the standard user-written SMI entry points for the Knight's Panel problem will be presented and described in detail. Each of the 22 routines (all of which are mandatory entry points) constructing the complete set of user-written routines are presented as a separate listing; lines are numbered for easy referencing in the text.

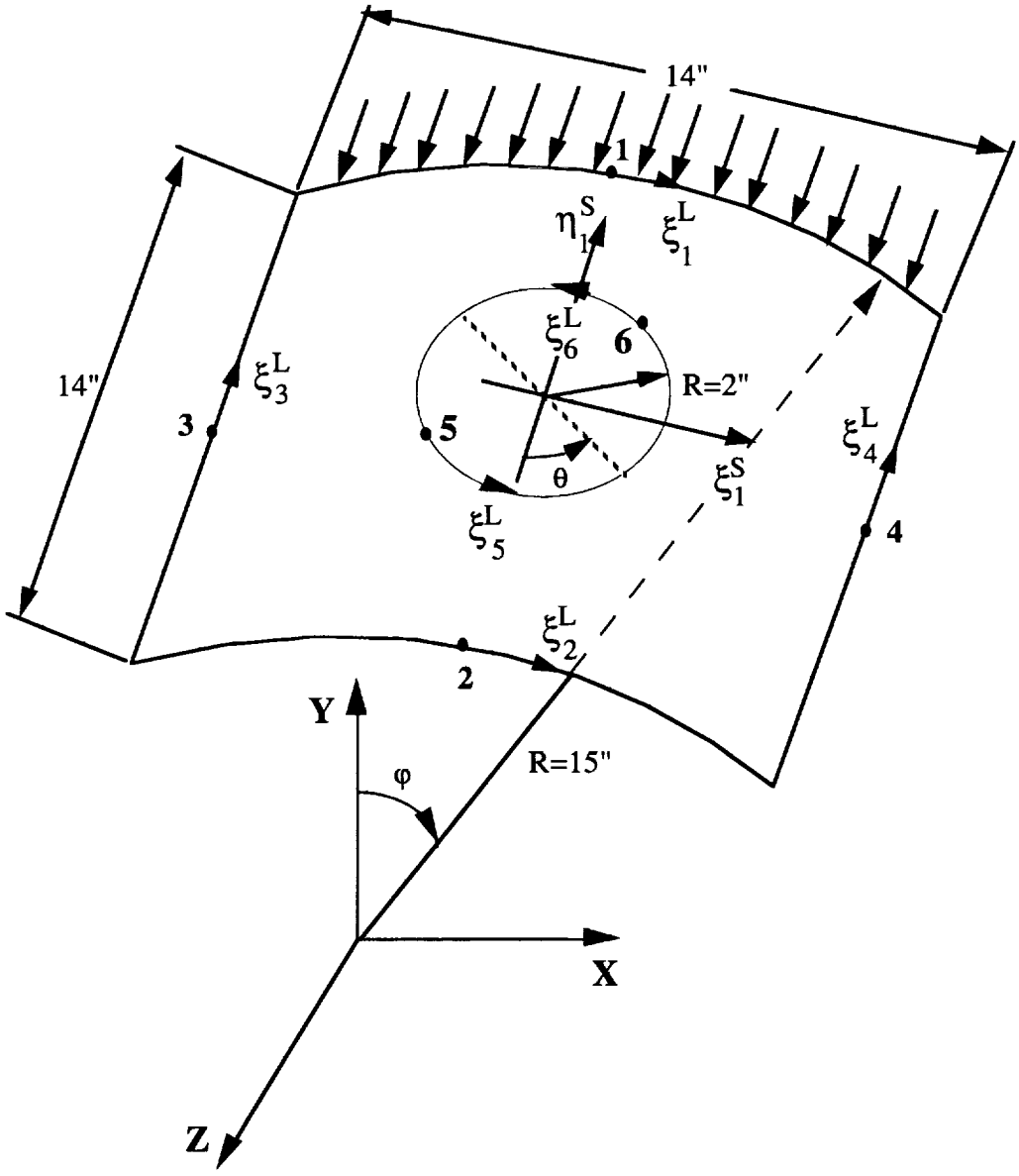


Figure 16.4-2 Knight's Panel: Problem Definition

16.4.4.1 Initialization Routine

The Initialization routine, shown in Listing 1, is merely a place holder for any initialization that may be required by the user-written routines or by a commercial solid modeling program (in the future). The user should implement this entry point, which is always the first routine to be called at the beginning of the refinement stage, to initialize any data or parameters required by particular user-written routines. For the current example, data initialization is not required and this routine has an empty executable body.

Listing 1 Initialization Routine

```
 1 c
 2 c     ***********
 3 c     S M I n i t
 4 c     ***********
 5 c
 6       subroutine SMInit ( status )
 7 c
 8 c.....Dummy initialization routine
 9 c
10 c
11 C ================================================================
12 c          D   e   c   l   a   r   a   t   i   o   n   s
13 C ================================================================
14 c
15 C=IF VAX
16 c
17       implicit none
18 c
19 C=ELSEIF SUN
20 c
21       implicit none
22 c
23 C=ELSE
24 c
25       implicit character *1 ( a - z )
26 c
27 C=ENDIF
28 c
29 c     High level DB utilities include file
30 c
31       include 'qsymbol.inc'
32 c
33       integer status
34 c
35 c ================================================================
36 c                        L   o   g   i   c
37 C ================================================================
38       if( status.ne.qOK ) return
39
40       return
41       end
```

16.4.4.2 Coordinates Routines

The geometry of the Knight's Panel example contains six geometry lines as depicted in Figure 16.4-2. Geometry lines 1 and 2 are circular arcs given by the following mathematical definitions:

$$Line1 = \left\{ \boldsymbol{x} \,|\, x = 15\cos(\varphi); y = 15\sin(\varphi); z = 0; \varphi \in [-\text{asin}\left(\frac{7}{15}\right), \ +\text{asin}\left(\frac{7}{15}\right)] \right\}$$

$$Line2 = \left\{ \boldsymbol{x} \,|\, x = 15\cos(\varphi); y = 15\sin(\varphi); z = 15; \varphi \in [-\text{asin}\left(\frac{7}{15}\right), \ +\text{asin}\left(\frac{7}{15}\right)] \right\}$$

The parametric presentations of these lines are obtained by replacing the variable φ by $\xi \in [-1,1]$ via:

$$\varphi = \xi \ \text{asin} \ \frac{7}{15}$$

Lines 53-62, and 36-72 of Listing 2 below contain the Fortran code representing the above equations.

Geometry lines 3 and 4 of Figure 16.4-2 are simple straight lines given by:

$$Line3 = \left\{ x | x = 15\cos(\bar{\varphi}); y = 15\sin(\bar{\varphi}); z = 7(1-\xi); \bar{\varphi} = -\text{asin}\left(\frac{7}{15}\right) \right\}$$

$$Line4 = \left\{ x | x = 15\cos(\bar{\varphi}); y = 15\sin(\bar{\varphi}); z = 7(1-\xi); \bar{\varphi} = \text{asin}\left(\frac{7}{15}\right) \right\}$$

and are programmed in lines 73-82 and 83-92 of Listing 2.

Finally, the circular hole at the center of the panel is a curve which defines the intersection of a cylinder of radius 2 inches along the y-axis with a cylinder of radius 15 inches along the z-axis. The mathematical presentation of this curve is:

$$Lines5, 6 = \{ x | x = 2\cos(\theta); z = 7 - 2\sin(\theta); y = \sqrt{15^2 - x^2} \}$$

In order to comply with restrictions 1 and 3 of "Limitations on User-Written SMI Option" above, this geometry line is divided into two segments, geometry lines 5 and 6, along the $\theta = -45^o$ diagonal. The mappings from the generic $\xi \in [-1,1]$ coordinate to the angle θ for each of these lines are given by:

$$\theta_5 = 90(1+\xi) + 225$$

$$\theta_6 = 90(1+\xi) + 45$$

This lines are programmed in lines 93-110 and 111-128 of Listing 2.

If the Line Coordinates routine is requested to provide information about a geometry line which is unknown to the routine, an error message is printed and error condition is set by calling ERR as shown in lines 129-139 of Listing 2.

Listing 2 Line Coordinates Routine

```
 1 c
 2 c     ***********
 3 c     L i n C r d
 4 c     ***********
 5 c
 6       subroutine LinCrd ( LineID , XSI , Crds , Status )
 7 c
 8 c.....Example solid model routine to compute global coordinates along
 9 c     a parametric presentation of a solid model line
10 c
11 C ***************************************************************
12 c          D   e   c   l   a   r   a   t   i   o   n   s
13 C ***************************************************************
14 c
15 C=IF VAX
16 c
17       implicit none
18 c
19 C=ELSEIF SUN
```

Listing 2 Line Coordinates Routine (Continued)

```
c
      implicit none
c
C=ELSE
c
      implicit character *1 ( a - z )
c
C=ENDIF
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
      integer LineID, Status
c
c=IF DOUBLE
c
      double precision
c
c=ELSE
c
      real
c
c=ENDIF
c
     $         XSI, Crds(3), Theta
c
c ======================================================================
c                        L    o    g    i    c
C ======================================================================
c
      if ( Status .ne. qOK ) return
c
      if ( LineID .eq. 1 ) then
c
c        Knight's panel geometry: line 1 - arc at Z=0.0
c
         Theta = XSI * asin ( 7.0 / 15.0 )
c
         Crds(1) = 15.0 * sin(Theta)
         Crds(2) = 15.0 * cos(Theta)
         Crds(3) = 0.0
c
      else if ( LineID .eq. 2 ) then
c
c        Knight's panel geometry: line 2 - arc at Z=14.0
c
         Theta = XSI * asin ( 7.0 / 15.0 )
c
         Crds(1) = 15.0 * sin(Theta)
         Crds(2) = 15.0 * cos(Theta)
         Crds(3) = 14.0
c
      else if ( LineID .eq. 3 ) then
c
c        Knight's panel geometry: line 3 -
                                  Straight line at Theta = - asin(7/15)
c
         Theta = - asin ( 7.0 / 15.0 )
c
         Crds(1) = 15.0 * sin(Theta)
         Crds(2) = 15.0 * cos(Theta)
         Crds(3) = 7.0 * ( XSI + 1.0 )
c
      else if ( LineID .eq. 4 ) then
c
c        Knight's panel geometry: line 4 -
                                  Straight line at Theta = + asin(7/15)
c
         Theta = + asin ( 7.0 / 15.0 )
c
         Crds(1) = 15.0 * sin(Theta)
         Crds(2) = 15.0 * cos(Theta)
         Crds(3) = 7.0 * ( XSI + 1.0 )
c
      else if ( LineID .eq. 5 ) then
c
```

Listing 2 Line Coordinates Routine (Continued)

```
 95 c        Knight's panel geometry: line 5 - circular cut-out
 96 c
 97 c          Note! X,Z are computed on the circle such that:
 98 c                      X * X + Z * Z = 4    (on the circle)
 99 c
100 c                Y is computed such that:
101 c                      X * X + Y * Y = 225 (on the panel)
102 c
103 c
104           Theta = ( ( XSI + 1.0 ) * 90.0 + 225.0 ) *
105      $            atan ( 1.0 ) / 45.0
106 c
107           Crds(1) = 2.0 * cos(Theta)
108           Crds(2) = sqrt ( 225.0 - Crds(1) * Crds(1) )
109           Crds(3) = 7.0 - 2.0 * sin(Theta)
110 c
111        else if ( LineID .eq. 6 ) then
112 c
113 c        Knight's panel geometry: line 6 - circular cut-out
114 c
115 c          Note! X,Z are computed on the circle such that:
116 c                      X * X + Z * Z = 4    (on the circle)
117 c
118 c                Y is computed such that:
119 c                      X * X + Y * Y = 225 (on the panel)
120 c
121 c
122           Theta = ( ( XSI + 1.0 ) * 90.0 + 45.0 ) *
123      $            atan ( 1.0 ) / 45.0
124 c
125           Crds(1) = 2.0 * cos(Theta)
126           Crds(2) = sqrt ( 225.0 - Crds(1) * Crds(1) )
127           Crds(3) = 7.0 - 2.0 * sin(Theta)
128 c
129        else
130 c
131 c        Unknown geometry line
132 c
133           call PRTs  ( qError , qIntegr ,
134      $                  '***ERROR*** unknown line geometry ID^'
135      $                //'            was requested in LinCrd^'
136      $                //'            unknown line ID :=' , LineID )
137 c
138           call ERR ( 'LinCrd' , 'unknown line ID' , Status )
139 c
140        end if
141 c
142        return
143        end
```

The Knight's Panel example contains a single surface defined by:

$$\mathrm{I} = \left\{ \boldsymbol{x} \mid x = 15\cos(\varphi); y = 15\sin(\varphi); z \in [0, 14]; \varphi \in [-\mathrm{asin}\left(\frac{7}{15}\right), \ + \mathrm{a}\right.$$

The surface generic coordinates, $\eta \in [-1,]$ are mapped to φ and z, respectively, via:

$$\varphi = \xi \mathrm{asin}\left(\frac{7}{15}\right)$$

$$z = 7(1 + \eta)$$

These surface definition equations are programmed in lines 53-61 of Listing 3. We do not have to account for the circular hole present in the surface since this hole is fully accounted for by the definition of the geometry lines (higher level entities in the goniometry hierarchy), and the

refinement processor will never request any information for points inside the hole since there are no elements defined inside this region in the initial mesh of the problem.

Listing 3 Surface Coordinates Routine

```
c
c     ***********
c     S u r C r d
c     ***********
c
      subroutine SurCrd ( SurfID , XSI , ETA , Crds , Status )
c
c.....Example solid model routine to compute global coordinates along
c     a parametric presentation of a solid model surface
c
C ================================================================
c          D   e   c   l   a   r   a   t   i   o   n   s
C ================================================================
c
C=IF VAX
c
      implicit none
c
C=ELSEIF SUN
c
      implicit none
c
C=ELSE
c
      implicit character *1 ( a - z )
c
C=ENDIF
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
      integer SurfID, Status
c
c=IF DOUBLE
c
      double precision
c
c=ELSE
c
      real
c
c=ENDIF
c
     $         XSI, ETA, Crds(3), Theta
c
c ================================================================
c                       L    o    g    i    c
C ================================================================
c
      if ( Status .ne. qOK ) return
c
      if ( SurfID .eq. 1 ) then
c
c        Knight's panel geometry
c
         Theta = XSI * asin ( 7.0 / 15.0 )
c
         Crds(1) = 15.0 * sin(Theta)
         Crds(2) = 15.0 * cos(Theta)
         Crds(3) = 7.0 * ( ETA + 1.0 )
c
      else
c
c        Unknown geometry surface surface
c
         call PRTs  ( qError , qIntegr ,
     $                '***ERROR*** unknown surface geometry ID^'
     $               //'            was requested in SurCrd^'
     $               //'            unknown surface ID :=' , SurfID )
c
         call ERR ( 'SurCrd' , 'unknown surface ID' , Status )
```

Listing 3 Surface Coordinates Routine (Continued)

```
c
      end if
c
      return
      end
```

The Volume Coordinates routine, shown in Listing 4, is an example of a typical dummy entry point in the user-written routines. This example does not require any volume definition; therefore, this routine contains only the formal Declarations section and an empty Logic section.

Listing 4 Volume Coordinates Routine

```
c
c     ***********
c     V o l C r d
c     ***********
c
      subroutine VolCrd ( VolmID , XSI , ETA , ZETA , Crds , Status )
c
c.....Dummy solid model routine
c
C ================================================================
c          D   e   c   l   a   r   a   t   i   o   n   s
C ================================================================
c
C=IF VAX
c
      implicit none
c
C=ELSEIF SUN
c
      implicit none
c
C=ELSE
c
      implicit character *1 ( a - z )
c
C=ENDIF
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
c=IF DOUBLE
c
      double precision
c
c=ELSE
c
      real
c
c=ENDIF
c
     $         XSI, ETA, ZETA, Crds(3)
c
      integer VolmID, Status
c
c ================================================================
c                      L   o   g   i   c
C ================================================================
c
      if ( Status .ne. qOK ) return
c
c     *************NOT REQUIRED FOR THIS MODEL**************
c
      return
      end
```

16.4.4.3 Boundary Condition Routines

The boundary conditions of the Knight's Panel example are summarized in "Initial Model Generation" above. These boundary codes are written in terms of standard Q-symbol parameters: "qSPCz" for specified zero DOF, "qSPCnz" for specified non-zero DOF, and "qFree" for free DOF. These boundary codes are defined for each of the six lines (see data statement in lines 36-41 of Listing 5) and for interior point along the surface (see data statement in line 36 of Listing 6).

The Logic part of these routines consists of merely copying the boundary codes associated with the requested line ID or surface ID from the internal storage arrays, "BCsi," to the receiving vector, "BCs" (see lines 49-52 of Listing 5 and lines 44-47 of Listing 6). Note the consistent error processing code present in this routines.

Listing 5 Line Boundary Conditions Routine

```
 1 c
 2 c     ***********
 3 c     L i n B c s
 4 c     ***********
 5 c
 6       subroutine LinBcs ( LineID , Bcs , NDof , Status )
 7 c
 8 c.....Example solid model routine to obtain boundary codes for lines
 9 c
10 C ================================================================
11 c          D   e   c   l   a   r   a   t   i   o   n   s
12 C ================================================================
13 c
14 C=IF VAX
15 c
16       implicit none
17 c
18 C=ELSEIF SUN
19 c
20       implicit none
21 c
22 C=ELSE
23 c
24       implicit character *1 ( a - z )
25 c
26 C=ENDIF
27 c
28 c     High level DB utilities include file
29 c
30       include 'qsymbol.inc'
31 c
32       integer LineID, NDof, Bcs(NDof), Status
33 c
34       integer Bcsi(6,6)
35 c
36       data Bcsi / qSPCz, qSPCz, qSPCnz, qSPCz, qSPCz, qSPCz,
37      $            qSPCz, qSPCz, qSPCz,  qSPCz, qSPCz, qSPCz,
38      $            qSPCz, qFree, qFree,  qSPCz, qFree, qSPCz,
39      $            qSPCz, qFree, qFree,  qSPCz, qFree, qSPCz,
40      $            qFree, qFree, qFree,  qFree, qFree, qFree,
41      $            qFree, qFree, qFree,  qFree, qFree, qFree/
42 c
43 c ================================================================
44 c                     L   o   g   i   c
45 C ================================================================
46 c
47       if ( Status .ne. qOK ) return
48 c
49       if ( LineID .gt. 0 .and. LineID .le. 6 ) then
50 c
51          call ICOPY ( Bcs , Bcsi(1,LineID) , NDof , Status )
52 c
53       else
54 c
55 c        Unknown geometry line
```

Listing 5 Line Boundary Conditions Routine (Continued)

```
c
          call PRTs  ( qError , qIntegr ,
     $                 '***ERROR*** unknown line geometry ID^'
     $               //'            was requested in LinBcs^'
     $               //'            unknown line ID :=' , LineID )
c
          call ERR ( 'LinBcs' , 'unknown line ID' , Status )
c
       end if
c
       return
       end
```

Listing 6 Surface Boundary Conditions Routine

```
c
c     ***********
c     S u r B c s
c     ***********
c
      subroutine SurBcs ( SurfID , Bcs , NDof , Status )
c
c.....Example solid model routine to obtain boundary codes for surfaces
c
C ================================================================
c          D   e   c   l   a   r   a   t   i   o   n   s
C ================================================================
c
C=IF VAX
c
      implicit none
c
C=ELSEIF SUN
c
      implicit none
c
C=ELSE
c
      implicit character *1 ( a - z )
c
C=ENDIF
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
      integer SurfID, NDof, Bcs(NDof), Status
c
      integer Bcsi(6)
c
      data Bcsi / qFree, qFree, qFree, qFree, qFree, qFree /
c
c ================================================================
c                      L    o    g    i    c
C ================================================================
c
      if ( Status .ne. qOK ) return
c
      if ( SurfID .eq. 1 ) then
c
         call ICOPY ( Bcs , Bcsi , NDof , Status )
c
      else
c
c        Unknown geometry surface
c
         call PRTs  ( qError , qIntegr ,
     $                '***ERROR*** unknown surface geometry ID^'
     $              //'            was requested in SurBcs^'
     $              //'            unknown surface ID :=' , SurfID )
c
         call ERR ( 'SurBcs' , 'unknown surface ID' , Status )
c
      end if
```

Listing 6 Surface Boundary Conditions Routine (Continued)

```
60 c
61       return
62       end
```

The Volume Boundary Conditions routine, shown in Listing 7, is a dummy entry point in the user-written routines. This example does not require any volume definitions; therefore, this routine contains only the formal Declarations section and an empty Logic section.

Listing 7 Volume Boundary Conditions Routine

```
 1 c
 2 c     ************
 3 c     V o l B c s
 4 c     ************
 5 c
 6       subroutine VolBcs ( VolmID , Bcs , NDof , Status )
 7 c
 8 c.....Example solid model routine to obtain boundary codes
 9 c     for Volums
10 c
11 C ================================================================
12 c           D   e   c   l   a   r   a   t   i   o   n   s
13 C ================================================================
14 c
15 C=IF VAX
16 c
17       implicit none
18 c
19 C=ELSEIF SUN
20 c
21       implicit none
22 c
23 C=ELSE
24 c
25       implicit character *1 ( a - z )
26 c
27 C=ENDIF
28 c
29 c     High level DB utilities include file
30 c
31       include 'qsymbol.inc'
32 c
33       integer VolmID, Status, NDof, Bcs(NDof), Status
34 c
35 c ================================================================
36 c                       L   o   g   i   c
37 C ================================================================
38 c
39       if ( Status .ne. qOK ) return
40 c
41 c     ******************NOT REQUIRED FOR THIS MODEL***************
42 c
43       return
44       end
```

16.4.4.4 Specified Displacements Routines

For the Knight's Panel example, the z-direction displacements along geometry line 1 are specified to be unity. The Specified Displacements Routines contain an additional logical argument, named "LExists," which is a flag indicating if the particular data, in this case nontrivial specified displacements, exists at the generic location. This information is required by the refinement processor in order to prevent the storage of trivial data in the database.

The Logic section of the "Line Specified Displacements" routine returns the value of the specified z-direction displacement in the "Spds" output vector for geometry line 1, sets the "LExists" flag to true and clears the output vector for all other lines (see lines 58-72 of Listing 8).

Listing 8 Line Specified Displacements Routine

```
 1 c
 2 c     ***********
 3 c     L i n S p d
 4 c     ***********
 5 c
 6       subroutine LinSpd ( LineID , XSI     , Spds   ,
 7      $                    NDof   , LExists , Status )
 8 c
 9 c.....Example solid model routine to obtain nodal specified
10 c     displacement vectors for lines
11 c
12 C ================================================================
13 c          D   e   c   l   a   r   a   t   i   o   n   s
14 C ================================================================
15 c
16 C=IF VAX
17 c
18       implicit none
19 c
20 C=ELSEIF SUN
21 c
22       implicit none
23 c
24 C=ELSE
25 c
26       implicit character *1 ( a - z )
27 c
28 C=ENDIF
29 c
30 c     High level DB utilities include file
31 c
32       include 'qsymbol.inc'
33 c
34       integer LineID, NDof, Status
35 c
36 c=IF DOUBLE
37 c
38       double precision
39 c
40 c=ELSE
41 c
42       real
43 c
44 c=ENDIF
45 c
46      $         XSI, Spds(NDof)
47 c
48       logical LExists
49 c
50 c ================================================================
51 c                       L   o   g   i   c
52 C ================================================================
53 c
54       if ( Status .ne. qOK ) return
55 c
56       LExists = .false.
57 c
58       if ( LineID .gt. 0 .and. LineID .le. 6 ) then
59 c
60          call RCLEAR ( Spds , NDof , Status )
61 c
62 c        Prescribed displacements in the computational 3-direction
63 c        along line # 1
64 c
65          if ( LineID .eq. 1 ) then
66
67             Spds(3) = 1.0
68
69             LExists = .true.
70
```

Listing 8 Line Specified Displacements Routine (Continued)

```
71          end if
72 c
73        else
74 c
75 c        Unknown geometry line
76 c
77          call PRTs  ( qError , qIntegr ,
78       $                 '***ERROR*** unknown line geometry ID^'
79       $               //'             was requested in LinSpd^'
80       $               //'             unknown line ID :=' , LineID )
81 c
82          call ERR ( 'LinSpd' , 'unknown line ID' , Status )
83 c
84        end if
85 c
86        return
87        end
```

The hierarchical nature of the geometry entities results in resolving all points for which non-trivial prescribed displacement data exists using their geometry line definition (e.g., all points along the geometry line 1). The Surface Specified Displacements routine is not required to provide any data for the requested points and it always sets the "LExists" flag to false, and clears the output vector "Spds" (see lines 56-61 of Listing 9).

Listing 9 Surface Specified Displacements Routine

```
 1 c
 2 c     ************
 3 c     S u r S p d
 4 c     ************
 5 c
 6       subroutine SurSpd ( SurfID , XSI  , ETA     ,
 7      $                    Spds   , NDof , LExists , Status )
 8 c
 9 c.....Example solid model routine to obtain nodal specified
10 c     displacement vectors for surfaces
11 c
12 C ================================================================
13 c          D   e   c   l   a   r   a   t   i   o   n   s
14 C ================================================================
15 c
16 C=IF VAX
17 c
18        implicit none
19 c
20 C=ELSEIF SUN
21 c
22        implicit none
23 c
24 C=ELSE
25 c
26        implicit character *1 ( a - z )
27 c
28 C=ENDIF
29 c
30 c     High level DB utilities include file
31 c
32        include 'qsymbol.inc'
33 c
34        integer SurfID, NDof, Status
35 c
36 c=IF DOUBLE
37 c
38        double precision
39 c
40 c=ELSE
41 c
42        real
43 c
44 c=ENDIF
45 c
46      $         XSI , ETA, Spds(NDof)
```

Listing 9 Surface Specified Displacements Routine (Continued)

```
c
      logical LExists
c
C ================================================================
c                        L   o   g   i   c
C ================================================================
c
      if ( Status .ne. qOK ) return
c
      LExists = .false.
c
      if ( SurfID .eq. 1 ) then
c
         call RCLEAR ( Spds , NDof , Status )
c
      else
c
         Unknown geometry surface
c
         call PRTs  ( qError , qIntegr ,
     $                '***ERROR*** unknown surface geometry ID^'
     $               //'            was requested in SurSpd^'
     $               //'            unknown surface ID :=' , SurfID )
c
         call ERR ( 'SurSpd' , 'unknown surface ID' , Status )
c
      end if
c
      return
      end
```

The Volume Specified Displacement routine, shown in Listing 10, is a dummy entry point in the user-written routines. This example does not require any volume definitions; therefore, this routine contains only the formal Declarations section and an empty Logic section.

Listing 10 Volume Specified Displacements Routine

```
c
c     ***********
c     V o l S p d
c     ***********
c
      subroutine VolSpd ( VolmID , XSI  , ETA     , ZETA   ,
     $                    Spds   , NDof , LExists , Status )
c
c.....Dummy solid model routine to obtain nodal specified
c     displacement vectors for volumes
c
C ================================================================
c          D   e   c   l   a   r   a   t   i   o   n   s
C ================================================================
c
C=IF VAX
c
      implicit none
c
C=ELSEIF SUN
c
      implicit none
c
C=ELSE
c
      implicit character *1 ( a - z )
c
C=ENDIF
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
      integer VolmID, Status, NDof, Status
c
```

Listing 10 Volume Specified Displacements Routine (Continued)

```
36 c=IF DOUBLE
37 c
38       double precision
39 c
40 c=ELSE
41 c
42       real
43 c
44 c=ENDIF
45 c
46      $          XSI, ETA, ZETA, Spds(NDof)
47 c
48       logical LExists
49 c
50 c ================================================================
51 c                          L    o    g    i    c
52 C ================================================================
53 c
54       if ( Status .ne. qOK ) return
55 c
56 c     ******************NOT REQUIRED FOR THIS MODEL***************
57 c
58       return
59       end
```

16.4.4.5 Computational Frame Transformations Routines

The Computational Frame Transformations routines define a transformation matrix from the global Cartesian coordinate system to a computational coordinate system. A 3x3 orthonormal transformation matrix is required at every nodal point of the model. The computational frame should be oriented for the user's convenience such that definitions of loads, boundary codes, and results (displacements, stresses, etc.) will be as simple and meaningful as possible. There are no restrictions on the choice of computational frames in COMET-AR and they may vary, in an arbitrary fashion, from one nodal point to another.

For the Knight's panel example, it is convenient to set the computational frame such that the first axis is normal to the panel, the third axis in the z-direction and the second axis completes the proper right-hand triad definition. For given point, $\boldsymbol{x}$, we can define the transformation from the global to computational frames, $\boldsymbol{T}_{cg}$, by:

$$\boldsymbol{T}_{cg} = \frac{1}{l}\begin{bmatrix} x & -y & 0 \\ y & x & 0 \\ 0 & 0 & l \end{bmatrix}^T$$

where

$$l = \sqrt{x^2 + y^2}$$

The Computational Frame Transformations routines are called with the generic point coordinate rather than the physical coordinates and they first have to call the appropriate Coordinates routine to obtain the global coordinates at the requested location (see lines 57-71 of Listing 11 for the geometry lines and again in lines 53-67 of Listing 12 for the geometry surface).

The transformation matrix, "Trfs," is programmed in lines 73-87 of Listing 11 for the geometry lines and again in lines 69-86 of Listing 12 for the geometry surface.

Listing 11 Line Computational Frame Transformations Routine

```
c
c     ***********
c     L i n T r f
c     ***********
c
      subroutine LinTrf ( LineID , XSI , Trfs , Status )
c
c.....Example solid model routine to obtain nodal transformations for lines
c
C ================================================================
c          D   e   c   l   a   r   a   t   i   o   n   s
C ================================================================
c
C=IF VAX
c
      implicit none
c
C=ELSEIF SUN
c
      implicit none
c
C=ELSE
c
      implicit character *1 ( a - z )
c
C=ENDIF
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
      integer LineID, Status
c
c=IF DOUBLE
c
      double precision
c
c=ELSE
c
      real
c
c=ENDIF
c
     $         XSI, Trfs(3,3), Crds(3), Vnorm
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
c ================================================================
c                        L   o   g   i   c
C ================================================================
c
      if ( Status .ne. qOK ) return
c
      call RCLEAR ( Trfs , 9 , Status )
c
c     Compute the global coordinates of the point
c
      call LinCrd ( LineID , XSI , Crds , Status )
c
      if ( Status .ne. qOK ) then
c
         call PRTs  ( qError , qIntegr ,
     $                '***ERROR*** Could not get point generic^'
     $               //'            coordinate from LinCrd in LinTrf^'
     $               //'            Line ID :=' , LineID )
c
         return
c
      end if
c
```

Listing 11 Line Computational Frame Transformations Routine (Continued)

```
      if ( LineID .gt. 0 .and. LineID .le. 6 ) then
c
c        Dir 1 - Normal to the panel
c        Dir 2 - tangent
c        Dir 3 - global Z
c
         call MultSV ( Trfs , 1.0 / Vnorm ( Crds , 2  , Status ) ,
     $                 Crds , 2 , Status )
c
         Trfs(1,2) = - Trfs(2,1)
         Trfs(2,2) =   Trfs(1,1)
         Trfs(3,3) =   1.0
c
         call Transpose ( Trfs , 3 , Status )
c
      else
c
c        Unknown geometry line
c
         call PRTs  ( qError , qIntegr ,
     $                '***ERROR*** unknown line geometry ID^'
     $               //'            was requested in LinTrf^'
     $               //'            unknown line ID :=' , LineID )
c
         call ERR ( 'LinTrf' , 'unknown line ID' , Status )
c
      end if
c
      return
      end
```

Listing 12 Surface Computational Frame Transformations Routine

```
c
c     ***********
c     S u r T r f
c     ***********
c
      subroutine SurTrf ( SurfID , XSI , ETA , Trfs , Status )
c
c.....Example solid model routine to obtain nodal transformations
      for surfaces
c
C ================================================================
c          D   e   c   l   a   r   a   t   i   o   n   s
C ================================================================
c
C=IF VAX
c
      implicit none
c
C=ELSEIF SUN
c
      implicit none
c
C=ELSE
c
      implicit character *1 ( a - z )
c
C=ENDIF
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
      integer SurfID, Status
c
c=IF DOUBLE
c
      double precision
c
c=ELSE
c
      real
```

Listing 12 Surface Computational Frame Transformations Routine (Continued)

```
c
c=ENDIF
c
      $         XSI, ETA, Trfs(3,3), Crds(3), Vnorm
c
c ========================================================================
c                        L    o    g    i    c
C ========================================================================
c
       if ( Status .ne. qOK ) return
c
       call RCLEAR ( Trfs , 9 , Status )
c
c      Compute the global coordinates of the point
c
       call SurCrd ( SurfID , XSI , ETA , Crds , Status )
c
       if ( Status .ne. qOK ) then
c
          call PRTs  ( qError , qIntegr ,
     $                 '***ERROR*** Could not get point generic^'
     $               //'            coordinates from SurCrd in SrfTrf^'
     $               //'            Surface ID :=' , SurfID )
c
          return
c
       end if
c
       if ( SurfID .eq. 1 ) then
c
       if ( Status .eq. qOK ) then
c
          Dir 1 - Normal to the panel
          Dir 2 - tangent
          Dir 3 - global Z
c
          call MultSV ( Trfs , 1.0 / Vnorm ( Crds , 2 , Status ) ,
     $                  Crds , 2 , Status )
c
c
          Trfs(1,2) = - Trfs(2,1)
          Trfs(2,2) =   Trfs(1,1)
          Trfs(3,3) =   1.0
c
          call Transpose ( Trfs , 3 , Status )
c
       else
c
          Unknown geometry surface
c
          call PRTs  ( qError , qIntegr ,
     $                 '***ERROR*** unknown surface geometry ID^'
     $               //'            was requested in SurTrf^'
     $               //'            unknown surface ID :=' , SurfID )
c
          call ERR ( 'SurTrf' , 'unknown surface ID' , Status )
c
       end if
c
       return
       end
```

The Volume Computational Frame Transformation routine, shown in Listing 13, is a dummy entry point in the user-written routines. This example does not require any volume definitions; therefore, this routine contains only the formal Declarations section and an empty Logic section.

Listing 13 Volume Computational Frame Transformation Routine

```
c
c     ***********
c     V o l T r f
c     ***********
```

Listing 13 Volume Computational Frame Transformation Routine (Continued)

```
 5 c
 6       subroutine VolTrf ( VolmID , XSI  , ETA    ,
 7      $                    ZETA   , Trfs , Status )
 8 c
 9 c.....Dummy solid model routine to obtain nodal transformations
10 c     for Volumes
11 c
12 C ================================================================
13 c          D   e   c   l   a   r   a   t   i   o   n   s
14 C ================================================================
15 c
16 C=IF VAX
17 c
18       implicit none
19 c
20 C=ELSEIF SUN
21 c
22       implicit none
23 c
24 C=ELSE
25 c
26       implicit character *1 ( a - z )
27 c
28 C=ENDIF
29 c
30 c     High level DB utilities include file
31 c
32       include 'qsymbol.inc'
33 c
34       integer VolmID, Status
35 c
36 c=IF DOUBLE
37 c
38       double precision
39 c
40 c=ELSE
41 c
42       real
43 c
44 c=ENDIF
45 c
46      $         XSI, ETA, ZETA, Trfs(3,3)
47 c
48 c ================================================================
49 c                        L   o   g   i   c
50 C ================================================================
51 c
52       if ( Status .ne. qOK ) return
53 c
54 c     ******************NOT REQUIRED FOR THIS MODEL**************
55 c
56       return
57       end
```

16.4.5 Material Fabrications Routines

The Material Fabrications routines establish the connection between a generic point location and the material properties of the point as stored by the Generic Constitutive Processor (GCP, see Chapter 8). These user-written routines can be used for generating heterogenous material data, thickness, and laminate variations and eccentricities changes within the model.

The Knight's Panel example consists of a single homogenous material throughout the model; therefore, the Material Fabrications routines return the same information for all of the geometry lines and the surfaces. These data consist of the fabrication identity number in the GCP database and dummy eccentricity data. The "FabRef" argument assigns the z-axis as the fabrication reference axis (i.e., the material frame's first coordinate direction is along the global z-axis). This

is programmed in lines 53-60 of Listing 14 for the geometry lines and in lines 53-59 of Listing 15 for the geometry surface.

Listing 14 Line Material Fabrication Routine

```
  1 c
  2 c     ***********
  3 c     L i n F a b
  4 c     ***********
  5 c
  6       subroutine LinFab ( LineID , XSI    , FabIDs , FabEcc ,
  7      $                    FabRef , NfabEc , Status           )
  8 c
  9 c.....Example solid model routine to obtain fabrications ID's for lines
 10 c
 11 C ================================================================
 12 c          D   e   c   l   a   r   a   t   i   o   n   s
 13 C ================================================================
 14 c
 15 C=IF VAX
 16 c
 17       implicit none
 18 c
 19 C=ELSEIF SUN
 20 c
 21       implicit none
 22 c
 23 C=ELSE
 24 c
 25       implicit character *1 ( a - z )
 26 c
 27 C=ENDIF
 28 c
 29 c     High level DB utilities include file
 30 c
 31       include 'qsymbol.inc'
 32 c
 33       integer LineID, FabIDs, FabRef, NfabEc, Status
 34 c
 35 c=IF DOUBLE
 36 c
 37       double precision
 38 c
 39 c=ELSE
 40 c
 41       real
 42 c
 43 c=ENDIF
 44 c
 45      $         XSI, FabEcc(*)
 46 c
 47 c ================================================================
 48 c                        L    o    g    i    c
 49 C ================================================================
 50 c
 51       if ( Status .ne. qOK ) return
 52 c
 53       if ( LineID .gt. 0 .and. LineID .le. 6 ) then
 54 c
 55          FabIDs    = 1
 56          NfabEc    = 1
 57          FabEcc(1) = 0.00
 58          FabRef    = 3
 59 c
 60       else
 61 c
 62 c        Unknown geometry line
 63 c
 64          call PRTs  ( qError , qIntegr ,
 65      $                '***ERROR*** unknown line geometry ID^'
 66      $               //'            was requested in LinFab^'
 67      $               //'            unknown line ID :=' , LineID )
 68 c
 69          call ERR ( 'LinFab' , 'unknown line ID' , Status )
 70 c
 71       end if
 72 c
```

Listing 14 Line Material Fabrication Routine (Continued)

```
      return
      end
```

Listing 15 Surface Material Fabrication Routine

```
c
c     ***********
c     S u r F a b
c     ***********
c
      subroutine SurFab ( SurfID , XSI    , ETA    , FabIDs ,
     $                    FabEcc , FabRef , NfabEc , Status )
c
c.....Example solid model routine to obtain fabrications ID's for surfaces
c
C ================================================================
c          D   e   c   l   a   r   a   t   i   o   n   s
C ================================================================
c
C=IF VAX
c
      implicit none
c
C=ELSEIF SUN
c
      implicit none
c
C=ELSE
c
      implicit character *1 ( a - z )
c
C=ENDIF
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
      integer SurfID, FabIDs, FabRef, NfabEc, Status
c
c=IF DOUBLE
c
      double precision
c
c=ELSE
c
      real
c
c=ENDIF
c
     $         XSI, ETA, FabEcc(*)
c
c ================================================================
c                      L   o   g   i   c
C ================================================================
c
      if ( Status .ne. qOK ) return
c
      if ( SurfID .eq. 1 ) then
c
         FabIDs    = 1
         NfabEc    = 1
         FabEcc(1) = 0.00
         FabRef    = 3
c
      else
c
c        Unknown geometry surface
c
         call PRTs  ( qError , qIntegr ,
     $                '***ERROR*** unknown surface geometry ID^'
     $               //'            was requested in SurFab^'
     $               //'            unknown surface ID :=' , SurfID )
c
         call ERR ( 'SurFab' , 'unknown surface ID' , Status )
```

Listing 15 Surface Material Fabrication Routine (Continued)

```
c
      end if
c
      return
      end
```

The Volume Material Fabrication routine, shown in Listing 16, is a dummy entry point in the user-written routines. This example dose not require any volume definitions; therefore, this routine contains only the formal Declarations section and an empty Logic section.

Listing 16 Volume Fabrication Routine

```
c
c     ***********
c     V o l F a b
c     ***********
c
      subroutine VolFab ( VolmID , XSI    , ETA    , ZETA   ,
     $                    FabIDs , FabEcc , FabRef , NfabEc ,
     $                    Status                            )
c
c.....Dummy solid model routine to obtain fabrications ID's for volumes
c
C ================================================================
c          D   e   c   l   a   r   a   t   i   o   n   s
C ================================================================
c
C=IF VAX
c
      implicit none
c
C=ELSEIF SUN
c
      implicit none
c
C=ELSE
c
      implicit character *1 ( a - z )
c
C=ENDIF
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
      integer VolmID, FabIDs, FabRef, NfabEc, Status
c
c=IF DOUBLE
c
      double precision
c
c=ELSE
c
      real
c
c=ENDIF
c
     $         XSI, ETA, ZETA, FabEcc(*)
c
c ================================================================
c                    L   o   g   i   c
C ================================================================
c
      if ( Status .ne. qOK ) return
c
c     ******************NOT REQUIRED FOR THIS MODEL**************
c
      return
      end
```

16.4.5.1 Body Load Routines

The Body Load routines are used to define a body load vector at each generic integration point. In general, these routines should be capable of defining a NDOF-dimensional body load vector, "Blds," at any given generic point along a geometry line, surface, or in a geometry volume.

A body load definition may range from the specification of a constant body load vector to an arbitrary spatially varying body load. In certain cases the global coordinate of a point may be required for the body load computation. It can be easily obtained by calling the appropriate Coordinates routine (see the *Computational Frame Transformations* section above for an example of such a call).

The Knight's Panel example does not include any body loads; therefore, the Body Loads routines simply set the "LExist" logical flag to false and clear the output load vector buffer as shown in lines 56-60 of Listing 17 for the geometry lines and in lines 56-62 of Listing 18 for the geometry surface.

Listing 17 Line Body Load Routine

```
 1 c
 2 c     ***********
 3 c     L i n B l d
 4 c     ***********
 5 c
 6       subroutine LinBld ( LineID , XSI     , Blds   ,
 7      $                    NDof   , LExists , Status )
 8 c
 9 c.....Example solid model routine to obtain body loads
10 c     vectors for lines
11 c
12 C ================================================================
13 c          D   e   c   l   a   r   a   t   i   o   n   s
14 C ================================================================
15 c
16 C=IF VAX
17 c
18       implicit none
19 c
20 C=ELSEIF SUN
21 c
22       implicit none
23 c
24 C=ELSE
25 c
26       implicit character *1 ( a - z )
27 c
28 C=ENDIF
29 c
30 c     High level DB utilities include file
31 c
32       include 'qsymbol.inc'
33 c
34       integer LineID, NDof, Status
35 c
36       logical LExists
37 c
38 c=IF DOUBLE
39 c
40       double precision
41 c
42 c=ELSE
43 c
44       real
45 c
46 c=ENDIF
47 c
48      $         XSI, Blds(NDof)
49 c
50 c ================================================================
```

Listing 17 Line Body Load Routine (Continued)

```
c                       L   o   g   i   c
C ======================================================================
c
      if ( Status .ne. qOK ) return
c
      if ( LineID .gt. 0 .and. LineID .le. 6 ) then
c
         LExists = .false.
         call RCLEAR ( Blds , NDof , Status )
c
      else
c
         Unknown geometry line
c
         call PRTs  ( qError , qIntegr ,
     $                '***ERROR*** unknown line geometry ID^'
     $               //'            was requested in LinBld^'
     $               //'            unknown line ID :=' , LineID )
c
         call ERR ( 'LinBld' , 'unknown line ID' , Status )
c
      end if
c
      return
      end
```

Listing 18 Surface Body Load Routine

```
c
c     ***********
c     S u r B l d
c     ***********
c
      subroutine SurBld ( SurfID , XSI  , ETA     ,
     $                    Blds   , NDof , LExists , Status )
c
c.....Example solid model routine to obtain body loads
c     vectors for surfaces
c
C ======================================================================
c          D   e   c   l   a   r   a   t   i   o   n   s
C ======================================================================
c
C=IF VAX
c
      implicit none
c
C=ELSEIF SUN
c
      implicit none
c
C=ELSE
c
      implicit character *1 ( a - z )
c
C=ENDIF
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
      integer SurfID, NDof, Status
c
      logical LExists
c
c=IF DOUBLE
c
      double precision
c
c=ELSE
c
      real
c
c=ENDIF
```

Listing 18 Surface Body Load Routine (Continued)

```
c
      $         XSI, ETA, Blds(NDof)
c
c ================================================================
c                        L   o   g   i   c
C ================================================================
c
       if ( Status .ne. qOK ) return
c
       if ( SurfID .eq. 1 ) then
c
          LExists = .false.
          call RCLEAR ( Blds , NDof , Status )
c
c
          Unknown geometry surface
c
          call PRTs  ( qError , qIntegr ,
      $                '***ERROR*** unknown surface geometry ID^'
      $               //'             was requested in SurBld^'
      $               //'             unknown surface ID :=' , SurfID )
c
          call ERR ( 'SurBld' , 'unknown surface ID' , Status )
c
       end if
c
       return
       end
```

The Volume Body Loads routine shown in Listing 19 below is a dummy entry point in the user-written routines. This example does not require any volume definitions; therefore, this routine contains only the formal Declarations section and an empty Logic section.

Listing 19 Volume Body Load Routine

```
c
c     ***********
c     V o l B l d
c     ***********
c
      subroutine VolBld ( VolmID , XSI  , ETA     , ZETA   ,
     $                    Blds   , NDof , LExists , Status )
c
c.....Dummy solid model routine to obtain body loads
c     vectors for volumes
c
C ================================================================
c          D   e   c   l   a   r   a   t   i   o   n   s
C ================================================================
c
C=IF VAX
c
      implicit none
c
C=ELSEIF SUN
c
      implicit none
c
C=ELSE
c
      implicit character *1 ( a - z )
c
C=ENDIF
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
      integer VolmID, Status, NDof
c
c=IF DOUBLE
c
```

Listing 19 Volume Body Load Routine (Continued)

```
      double precision
c
c=ELSE
c
      real
c
c=ENDIF
c
     $         XSI, ETA, ZETA, Blds(NDof)
c
      logical LExists
c
c ================================================================
c                        L   o   g   i   c
C ================================================================
c
      if ( Status .ne. qOK ) return
c
c     ******************NOT REQUIRED FOR THIS MODEL**************
c
      return
      end
```

16.4.5.2 Surface Load Routines

The Surface Pressure Load routine is used to define a pressure load at each generic integration point along a surface. A pressure load definition may range from the specification of a uniform pressure load to an arbitrary spatially varying one. In certain cases the global coordinate of point may be required for the pressure load computation. It can be easily obtained by calling the appropriate Coordinates routine (see the Computational Frame Transformations section above for an example of such a call).

The Knight's Panel example does not contain any pressure loads; therefore, the Surface Pressure Loads routine simply sets the "LExist" logical flag to false and clears the output load variable as shown in lines 56-60 of Listing 20.

Listing 20 Surface Pressure Routine

```
c
c     ***********
c     S u r P l d
c     ***********
c
      subroutine SurPld ( SurfID , XSI     , ETA    ,
     $                    Plds   , LExists , Status )
c
c.....Example solid model routine to obtain pressure loads
c     vectors for surfaces
c
C ================================================================
c          D   e   c   l   a   r   a   t   i   o   n   s
C ================================================================
c
C=IF VAX
c
      implicit none
c
C=ELSEIF SUN
c
      implicit none
c
C=ELSE
c
      implicit character *1 ( a - z )
c
```

Listing 20 Surface Pressure Routine (Continued)

```
C=ENDIF
c
c     High level DB utilities include file
c
      include 'qsymbol.inc'
c
      integer SurfID, Status
c
      logical LExists
c
c=IF DOUBLE
c
      double precision
c
c=ELSE
c
      real
c
c=ENDIF
c
     $         XSI, ETA, Plds
c
c ================================================================
c                        L    o    g    i    c
C ================================================================
c
      if ( Status .ne. qOK ) return
c
      if ( SurfID .eq. 1 ) then
c
         LExists = .false.
         Plds = 0.0
c
      else
c
         Unknown geometry surface
c
         call PRTs  ( qError , qIntegr ,
     $                '***ERROR*** unknown surface geometry ID^'
     $              //'             was requested in SurPld^'
     $              //'             unknown surface ID :=' , SurfID )
c
         call ERR ( 'SurPld' , 'unknown surface ID' , Status )
c
      end if
c
      return
      end
```

The Surface Traction Load routines are used to define a surface traction load vector at each generic nodal point location. In general, these routines should be capable of defining an NDOF-dimensional surface traction vector, "Slds," at any given generic point along a geometry surface.

A surface traction load definition may range from the specification of a constant surface traction load vector to an arbitrary spatially varying load. In certain cases the global coordinate of a point may be required for the surface traction load computation. It can be easily obtained by calling the appropriate Coordinates routine (see the Computational Frame Transformations section above for an example of such a call).

The Knight's Panel example does not include any surface traction loads; therefore, the Surface Traction Loads routines simply set the "LExist" logical flag to false and clear the output load vector buffer as shown in lines 56-60 of Listing 21.

Listing 21 Surface (Traction) Loads Routine

```
 1 c
 2 c     ***********
 3 c     S u r S l d
 4 c     ***********
 5 c
 6       subroutine SurSld ( SurfID , XSI  , ETA     ,
 7      $                    Slds   , NDof , LExists , Status )
 8 c
 9 c.....Example solid model routine to obtain surface (traction)
10 c     loads vectors for surfaces
11 c
12 C ================================================================
13 c          D   e   c   l   a   r   a   t   i   o   n   s
14 C ================================================================
15 c
16 C=IF VAX
17 c
18       implicit none
19 c
20 C=ELSEIF SUN
21 c
22       implicit none
23 c
24 C=ELSE
25 c
26       implicit character *1 ( a - z )
27 c
28 C=ENDIF
29 c
30 c     High level DB utilities include file
31 c
32       include 'qsymbol.inc'
33 c
34       integer SurfID, NDof, Status
35 c
36       logical LExists
37 c
38 c=IF DOUBLE
39 c
40       double precision
41 c
42 c=ELSE
43 c
44       real
45 c
46 c=ENDIF
47 c
48      $         XSI, ETA, Slds(NDof)
49 c
50 c ================================================================
51 c                      L   o   g   i   c
52 C ================================================================
53 c
54       if ( Status .ne. qOK ) return
55 c
56       if ( SurfID .eq. 1 ) then
57 c
58          LExists = .false.
59          call RCLEAR ( Slds , NDof , Status )
60 c
61       else
62 c
63 c        Unknown geometry surface
64 c
65          call PRTs  ( qError , qIntegr ,
66      $                '***ERROR*** unknown surface geometry ID^'
67      $              //'            was requested in SurSld^'
68      $              //'            unknown surface ID :=' , SurfID )
69 c
70          call ERR ( 'SurSld' , 'unknown surface ID' , Status )
```

Listing 21 Surface (Traction) Loads Routine (Continued)

```
 71 c
 72       end if
 73 c
 74       return
 75       end
```

16.4.5.3 Line Load Routine

The Line Loads routines are used to define a Line load vector at each generic nodal point location. In general, these routines should be capable of defining an NDOF-dimensional line load vector, "Llds," at any given generic point along a geometry line.

A line load definition may range from the specification of a constant line traction vector to an arbitrary spatially varying load. In certain cases the global coordinate of a point may be required for the line load computation. It can be easily obtained by calling the appropriate Coordinates routine (see the *Computational Frame Transformations* section above for an example of such a call).

The Knight's Panel example does not include any line loads; therefore, the Line Loads routine simply sets the "LExist" logical flag to false and clears the output load vector buffer as shown in lines 56-60 of Listing 22.

Listing 22 Line Load Routine

```
  1 c
  2 c     ***********
  3 c     L i n L l d
  4 c     ***********
  5 c
  6       subroutine LinLld ( LineID , XSI     , Llds   ,
  7      $                    NDof   , LExists , Status )
  8 c
  9 c.....Example solid model routine to obtain line loads
 10 c     vectors for lines
 11 c
 12 C ================================================================
 13 c          D   e   c   l   a   r   a   t   i   o   n   s
 14 C ================================================================
 15 c
 16 C=IF VAX
 17 c
 18       implicit none
 19 c
 20 C=ELSEIF SUN
 21 c
 22       implicit none
 23 c
 24 C=ELSE
 25 c
 26       implicit character *1 ( a - z )
 27 c
 28 C=ENDIF
 29 c
 30 c     High level DB utilities include file
 31 c
 32       include 'qsymbol.inc'
 33 c
 34       integer LineID, NDof, Status
 35 c
```

Listing 22 Line Load Routine (Continued)

```
      logical LExists
c
c=IF DOUBLE
c
      double precision
c
c=ELSE
c
      real
c
c=ENDIF
c
     $         XSI, Llds(NDof)
c
c ================================================================
c                        L    o    g    i    c
C ================================================================
c
      if ( Status .ne. qOK ) return
c
      LExists = .false.
c
      if ( LineID .gt. 0 .and. LineID .le. 6 ) then
c
         call RCLEAR ( Llds , NDof , Status )
c
      else
c
         Unknown geometry line
c
         call PRTs  ( qError , qIntegr ,
     $                '***ERROR*** unknown line geometry ID^'
     $               //'            was requested in LinLld^'
     $               //'            unknown line ID :=' , LineID )
c
         call ERR ( 'LinLld' , 'unknown line ID' , Status )
c
      end if
c
      return
      end
```

16.4.6 References

[1] Stanley, G. and Swenson, L., *HDB: Object-Oriented Database Utilities for COMET-AR*, NASA Computational Structural Mechanics (CSM) Contract Report, August 1992.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1997	Contractor Report

4. TITLE AND SUBTITLE
COMET-AR User's Manual
COmputational MEchanics Testbed with Adaptive Refinement

5. FUNDING NUMBERS
L-44380D
505-63-53-01

6. AUTHOR(S)
E. Moas, Editor

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Applied Research Associates, Inc.
Southeast Division
811 Spring Forest Road, Suite 100
Raleigh, NC 27609

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
NASA Langley Research Center
Hampton, VA 23681-2199

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA/CR-97-206284

11. SUPPLEMENTARY NOTES
Prepared for Langley Research Center under Air Force Contract F08635-93-C
Langley Technical Monitor: Jonathan B. Ransom

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Category 39 Distribution: Nonstandard
Availability: NASA CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)
The COMET-AR User's Manual provides a reference manual for the Computational Structural Mechanics Testbed with Adaptive Refinement (COMET-AR), a software system developed jointly by Lockheed Palo Alto Research Laboratory and NASA Langley Research Center under contract NAS1-18444. The COMET-AR system is an extended version of an earlier finite element based structural analysis system called COMET, also developed by Lockheed and NASA. The primary extensions are the adaptive mesh refinement capabilities and a new "object-like" database interface that makes COMET-AR easier to extend further. This User's Manual provides a detailed description of the user interface to COMET-AR from the viewpoint of a structural analyst.

14. SUBJECT TERMS
Computational Structural Mechanics
Structural Analysis

15. NUMBER OF PAGES
895

16. PRICE CODE
A99

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	